8x93x Family Universal Serial Bus Evaluation Board Manual

May 1997

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature, may be obtained from:

Intel Corporation
P.O. Box 7641
Mt. Prospect, IL 60056-7641

or call 1-800-879-4683

FIGURES

TABLES

1

Guide to This Manual

CHAPTER 1
GUIDE TO THIS MANUAL

This manual describes the 8x930 Family Universal Serial Bus (USB) evaluation board and how to operate it. This board is designed for initial evaluation and development of 8x93x code. Please read this manual before powering up the board.

The board arrives with the 8x930Ax embedded microcontroller installed on it. The board also supports adapter boards that are populated with either the 8x930 or 8x931 Family embedded microcontrollers. The 8x931 Family microcontrollers use the standard instruction set of the MCS® 51 architecture. The 8x930 Family microcontrollers use the standard instruction set of the MCS® 251 architecture, which is binary code compatible with the MCS® 51 architecture.

The evaluation board is based on the 8x930 Family USB microcontroller. The microcontroller consist of standard 8XC251 peripherals and a USB module. The module contains one upstream and four downstream ports. It integrates the USB transceivers, serial bus interface engine (SIE), function interface unit (FIU), and transmit/receive FIFOs. Both the evaluation board and the microcontrollers are fully compliant with *Universal Serial Bus Specification*, Rev. 1.

You can use this board as part of your prototype application hardware during design and development. A Reduced Instruction Set Monitor (RISM) is included in an EPROM (or Flash memory) device which can communicate with compatible personal-computer resident debuggers or monitors.

1.1 AUDIENCE

This manual was written for design engineers who are familiar with microcontrollers.

1.2 MANUAL CONTENTS

This manual has six chapters and three appendixes.

This chapter provides an overview of the manual. It summarizes the contents of the remaining chapters and the appendix. It also describes notational conventions and terminology; lists related documents, products, data sheets, and user manual supplements; and gives important numbers for obtaining application support.

Chapter 2, “Getting Started”—lists the contents of the development kit, discusses hardware and software requirements, and gives the default board settings. It also explains how to interface the board with the debugger and how to use the board in stand-alone mode.

Chapter 3, “Hardware and Functional Overview”—describes the functional units of the evaluation board. It includes a block diagram of the board and a diagram of the board’s major components. It also discusses the features of the 8x930Ax and adapter board expandability.

Chapter 4, “Memory Configuration”—discusses code space, data space, and configuration bytes.

Chapter 5, “RISM”—describes RISM, how it is configured for this evaluation board, its resources, and its commands and command usage. It also describes how to implement steps and breakpoints and describes the USB reset sequence.

Chapter 6, “Jumper and DIP-Switch Settings”—lists and describes the jumper and DIP-switch settings for the 8x930 Family USB evaluation board.

Appendix A, “Components and Connectors”—contains a diagram of the evaluation board showing the board’s major components and connectors, a table listing all components, another table describing the serial-port connector and the cable that connects the evaluation board to the host PC, and a figure showing how to assemble a 25-pin to 9-pin serial-port, interface adapter cable to attach to the evaluation board.

Appendix B, “CPLD Equations”—includes CPLD code.

Appendix C, “Schematics”—includes schematics of the 8x930 Family USB evaluation board.

1.3 NOTATIONAL CONVENTIONS AND TERMINOLOGY

The following notations and terminology are used throughout this manual.

— The pound symbol (#) has two meanings, depending on the context. When used with a signal name, it indicates that the signal is active low. When used in an instruction, the symbol prefixes an immediate value in immediate addressing mode.

italics — Italics identify variables and introduce new terminology. The context in which italics are used distinguishes between the two possible meanings.

Variables in registers and signal names are commonly represented by *x* and *y*, where *x* represents the first variable and *y* represents the second variable. For example, in register P*x*_MODE.*y*, *x* represents the variable that identifies the specific port, and *y* represents the register bit variable [7:0 or 15:0].

X Uppercase X (no italics) represents an unknown value or a "don't care" state or condition. The value may be either binary or hexadecimal, depending upon the context. For example, the hexadecimal value FF2XAFH indicates that bits 11:8 are unknown; 10XX in binary context indicates that the two least significant bits (LSBs) are unknown.

Board Components The following abbreviations are used to represent discrete and active components.

C*x*	capacitor
D*x*	diode
DP*x*	LED bank
E*x*	jumper
JP*x*	connector
L*x*	inductor
P*x*	port
R*x*	resistor
RP*x*	resistor pack
S*x*	switch
U*x*	device (e.g., latch, buffer, memory, controller)

Assert and Deassert The terms *assert* and *deassert* refer to the act of making a signal active (enabled) and inactive (disabled), respectively. The active polarity (high/low) is defined by the signal name. Active-low signals are designated by a pound symbol (#) suffix; active-high signals have no suffix. To assert RD# is to drive it low (equal to or less than V_{OL}); to assert ALE is to drive it high (equal to or greater than V_{OH}); to deassert RD# is to drive it high; to deassert ALE is to drive it low.

Instructions Instruction mnemonics are shown in upper case; however, you may use either upper case or lower case in your source code.

NC The term "NC" is an abbreviation for "no connection." It indicates that no connection is required.

Numbers Hexadecimal numbers are represented by a string of hexadecimal digits followed by the character H. Decimal and binary numbers are represented by their customary notations. (That is, 255 is a decimal number and 11111111 is a binary number. In some cases, the letter B is appended to binary numbers for clarity.

Units of Measure

The following abbreviations are used to represent units of measure:

A	amps, amperes
mA	milliamps, milliamperes
Kbyte	kilobytes
KHz	kilohertz
KΩ	kilo-ohms
Mbyte	megabytes
MHz	megahertz
ms	milliseconds
mW	milliwatts
ns	nanoseconds
pF	picofarads
V	voltage, volts
VDC	voltage, direct current
VAC	voltage, alternating current
W	watts
µA	microamps, microamperes
µF	microfarads
µs	microseconds

Register Bits

Bit locations are indexed by 7:0 (or 15:0), where bit 0 is the least-significant bit and 7 (or 15) is the most-significant bit. An individual bit is represented by the register name, followed by a period and the bit number. For example, WSR.7 is bit 7 of the window select register. In some discussions, bit names are used. For example, the name of WSR.7 is HLDEN.

Register Names

Register names are shown in upper case. For example, TIMER2 is the timer 2 register; timer 2 is the timer. If a register name contains a lowercase character, it represents more than one register. For example, P*x*_REG represents four registers: P1_REG, P2_REG, P3_REG, and P4_REG.

Reserved Bits

Certain bits are described as *reserved* bits. In illustrations, reserved bits are indicated with a dash (—). These bits are not used in this device and may be used in future implementations. To help ensure that a current software design is compatible with future implementations, reserved bits should be cleared (given a value of “0”), unless otherwise noted.

Set and Clear The terms *set* and *clear* refer to the value of a bit or the act of giving it a value. When a bit is *set*, its value is "1"; *setting* a bit gives it a "1" value. When a bit is *clear*, its value is "0"; *clearing* a bit gives it a "0" value.

Signal Names Signal names are shown in upper case. When several signals share a common name, an individual signal is represented by the signal name followed by a number. For example, the EPA signals are named EPA0, EPA1, EPA2, etc. Port pins are represented by the port abbreviation, a period, and the pin number (e.g., P1.0, P1.1). A pound symbol (#) appended to a signal name identifies an active-low signal.

1.4 RELATED DOCUMENTS

Table 1-1 lists the names of documents that are useful in designing systems using an 8x930 Family embedded microcontroller. To order these documents, contact Intel Literature Fulfillment via one of the following methods:

- U.S. and Canada: phone 1-800-548-4725
- Europe: phone +44(0) 793-431155
- Internet users: visit the Intel World Wide Web site (http://www.intel.com).

Table 1-1. Related Documents

Document Name	Order Number
8x930Ax, 8x930Hx Universal Serial Bus Microcontroller User's Manual	272949
Embedded Microcontrollers	270646
Embedded Processors	272396
Embedded Applications	270648
Packaging	240800
Universal Serial Bus Specification	272904

1.4.1 Datasheets

Embedded Microcontrollers includes a datasheet for each product. The datasheets are listed in Table 1-2. You can also order each datasheet individually.

Table 1-2. Datasheets

Datasheet	Order Number
8x930Ax Universal Serial Bus Microcontroller	272917

1.4.2 Application Notes

The MCS 251 application notes listed in Table 1-3 apply to the 8x930 Family microcontrollers.

Table 1-3. MCS 251 Application Notes

Application Note	Order Number
AP-125, Designing Microcontroller Systems for Electrically Noisy Environments	210313
AP-155, Oscillators for Microcontrollers	230659
AP-708, Introducing the MCS® 251 Microcontroller—the 8XC251SB	272670
AP-709, Maximizing Performance Using MCS® 251 Microcontroller—Programming the 8XC251SB	272671
AP-710, Migrating from the MCS® 51 Microcontroller to the MCS 251 Microcontroller (8XC251SB)—Firmware and Hardware Considerations	272672

The MCS 51 application notes listed in Table 1-4 apply to the 8x931 Family microcontrollers.

Table 1-4. MCS 51 Application Notes

Application Note	Order Number
AP70, Using the Intel MCS® 51 Boolean Processing Capabilities	203830
AP-223, 8051 Based CRT Terminal Controller	270032
AP-252, Designing With the 80C51BH	270068
AP-425, Small DC Motor Control	270622
AP-410, Enhanced Serial Port on the 83C51FA	270490
AP-415, 83C51FA/FB PCA Cookbook	270609
AP-476, How to Implement I^2C Serial Communication Using Intel MCS® 51 Microcontrollers	272319

1.5 APPLICATION SUPPORT SERVICES

You can get up-to-date technical information from a variety of electronic support systems: the World Wide Web, the FaxBack* service, and Intel's Brand Products and Applications Support bulletin board service (BBS). These systems are available 24 hours a day, 7 days a week.

In the U.S. and Canada, technical support representatives are available between 5 a.m. and 5 p.m. Pacific Standard Time (PST). Outside the U.S. and Canada, please contact your local distributor. You can order product literature from Intel literature centers and sales offices.

Table 1-5 lists information for accessing these services.

Table 1-5. Intel Support Services

Service	U.S. and Canada	Asia-Pacific and Japan	Europe
World Wide Web	http://www.intel.com/	†http://www.intel.com/	†http://www.intel.com/
FaxBack*	1-800-525-3019	1-503-264-6835 1-916-356-3105	+44(0)1793-496646
BBS	1-503-264-7999 1-916-356-3600	1-503-264-7999 1-916-356-3600	+44(0)1793-432955
Help Desk	1-800-628-8686 1-916-356-7999	Please contact your local distributor.	Please contact your local distributor.
Literature	1-800-548-4725	1-708-296-9333 +81(0)120 47 88 32	+44(0)1793-431155 England +44(0)1793-421777 France +44(0)1793-421333 Germany

† A selector box at the bottom of this web site allows you to select a language in which you would like to view the web site.

1.5.1 World Wide Web (WWW)

You can find the following information on the World Wide Web:

- This document, its updates, and addendums at:
  http://developer.intel.com/design/usb/manuals
- A variety of product information at:
  http://developer.intel.com/design/usb
- USB software support at:
  http://developer.intel.com/design/usb/swsup
- Information about USB and specifications at:
  http://www.usb.org
- Intel financial information, history, and news at:
  http://www.intel.com

1.5.2 FaxBack Service

FaxBack is an on-demand publishing system that sends documents to your fax machine. You can get product announcements, change notifications, product literature, device characteristics, design recommendations, and quality and reliability information from FaxBack 24 hours a day, 7 days a week.

Think of the FaxBack service as a library of technical documents you can access with your phone. Just dial the FaxBack telephone number listed in Table 1-5 and respond to the system prompts. After you select a document, the system sends a copy to your fax machine.

Each document has an order number and is listed in a subject catalog. The first time you use FaxBack, order the appropriate subject catalogs to get a complete listing of document order numbers. Catalogs are updated regularly, so call for the latest information.

1.5.3 Bulletin Board Service

The bulletin board system (BBS) lets you download files to your computer. The application BBS has the latest *Ap*BUILDER software, hypertext manuals and datasheets, software drivers, firmware upgrades, application notes and utilities, and quality and reliability data.

Any customer with a modem and computer can access the BBS. The system provides automatic configuration support for 1200- through 19200-baud modems. Use these modem settings: no parity, 8 data bits, and 1 stop bit.

To access the BBS, just dial the telephone number (see Table 1-5 on page 1-7) and respond to the system prompts. During your first session, the system asks you to register with the system operator by entering your name and location. The system operator will set up your access account within 24 hours. After this time, you can access the files on the BBS.

NOTE

In the U.S. and Canada, you can get a BBS user's guide, a master list of BBS files, and a list of FaxBack documents by calling 1-800-525-3019. Use these modem settings: no parity, 8 data bits, and 1 stop bit.

2

Getting Started

CHAPTER 2
GETTING STARTED

The 8x930 Family USB evaluation board is part of the USB development kit. Use this evaluation board for code development and debugging. You can also use this board in stand-alone mode, as a function, or as a hub. (To use the evaluation board as a hub, you need to obtain the appropriate adapter board kit. The adapter boards can only be used with the 4 port version of the 8x930 Family USB evaluation board.) The EPROM shipped on the board contains four versions of the Intel Reduced Instruction Set Monitor (RISM); it also contains the 8x930Ax firmware.

This chapter lists the contents of the development kit as well as hardware and software requirements. It also discusses the board's features, default settings, and use in stand-alone and debugging modes.

2.1 DEVELOPMENT KIT CONTENTS

The 8x930 Family USB development kit includes the following items.

- 8x930 Family USB evaluation board, which includes the following features (see Figure 2-1 on page 2-3):
  - — 8x930Ax microcontroller installed on the board.
  - — One upstream USB port.
  - — Four downstream USB ports.†
  - — Two serial ports (RS-232).
  - — 32-Kbyte EPROM programmed with four RISM modes for code downloading and debugging and firmware for the 8x930Ax microcontroller.
  - — Ability to support the following modes: binary page/nonpage and source page/nonpage modes.
  - — 128-Kbyte SRAM for code and data.
  - — Self- and USB-bus powered options.
  - — Power management circuitry.
  - — LEDs connected to Port 1 to facilitate debugging.
- A USB A-to-B cable.
- A DB-9S, RS-232, 9-pin straight-through cable.
- Third-party vendor software (evaluation versions).
- *Ap*BUILDER Interactive programming software.
- Various documents about the device and development tools.
- Technical Documentation: The development kit includes this manual, the *8x93x Family Universal Serial Bus Evaluation Board Manual*. For available related documentation, refer to “Related Documents” on page 1-5. Appendix C, “Schematics” includes a set of evaluation board schematics.

† NOTE: To enable the capability of four downstream ports, you will need to obtain a hub function adapter board kit. This adapter board may only be used with the 4 port 8*x*930 Family USB evaluation board.

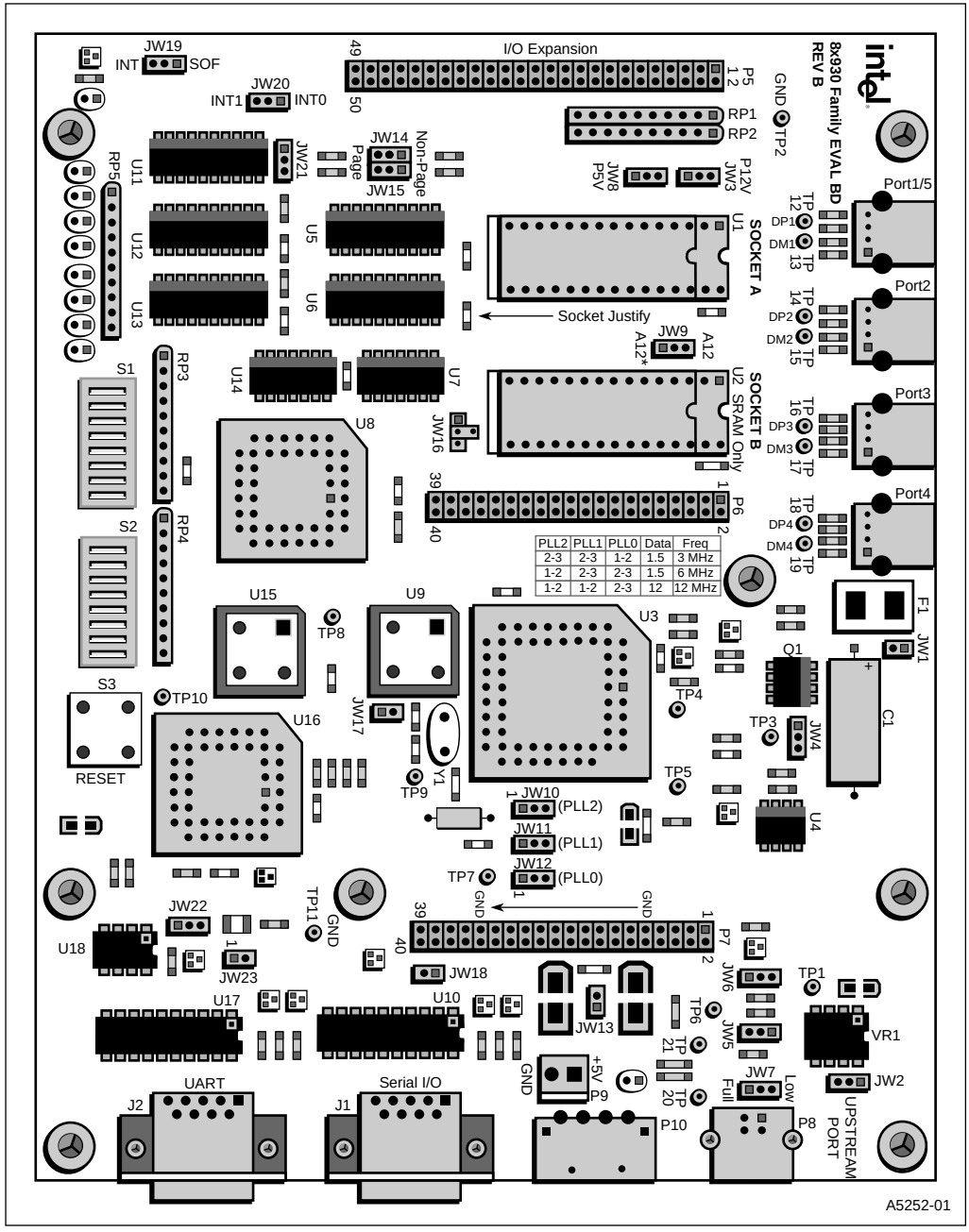


Figure 2-1. 8x930 Family USB Evaluation Board Layout

2.2 HARDWARE REQUIREMENTS

Use the 8x930 Family USB evaluation board as a stand-alone unit or interface it with a personal computer (PC) to facilitate debugging and ease of operation.

You need the following hardware to generate, assemble, and debug code on this evaluation board.

- An IBM PC or compatible, 386 or higher, with at least one available COM port. The evaluation board can interface with the host PC through either com1 or com2. Use either the 8x930 on-board serial port or the external UART port for this communication.
- A USB A-to-B cable.
- A DB-9S, RS-232 cable with a 9-pin connector at the end that connects the evaluation board to a PC COM port.
- A regulated + 5 volt power supply and +12 volt if you are going to program flash.

2.3 SOFTWARE REQUIREMENTS

The evaluation board can be interfaced with a software environment for code development and debugging. Write all code in assembly language, C, or a combination of both languages.

The development kit includes evaluation versions of third-party software needed to develop code for this evaluation board. This software can be interfaced with the board through a RS-232 serial link from a personal computer on which the application code is being developed. You can also download executable programs to the board.

Choose a software package and install it. Each package includes a debugger environment that can be interfaced with the RISM programmed on the board's EPROM. This RISM allows you to download code to the SRAM on the board and facilitates debugging.

NOTE

Developer code must start at location 00:4000h if you plan to use RISM. This location is hard-coded in RISM. If users start application code at another location, RISM will not function properly and your code will not run.

2.4 CONNECTING THE EVALUATION BOARD TO THE HOST SYSTEM

Complete the following procedure to connect the 8x930 Family USB evaluation board to the host system and power up the evaluation board (Figure 2-1 on page 2-3 shows the location of the evaluation board's power, ground, and serial port connections.):

1. Turn off power to the PC and the power supply.
2. Connect the serial port cable (the DB-9S, RS-232 cable) from the board's J2 connector, labeled UART, to either the com1 or com2 serial port on your PC.

   (See Table A-2 on page A-5 for details about the individual components of the DB-9S, RS-232 cable.)
3. Connect the power cable from the power supply to either the P9 or P10 power connector on the evaluation board.

WARNING

You **must** use a +5 volt-regulated power supply. Lower voltage may not operate the evaluation board. Higher voltage may damage the board. An unregulated power supply may cause unpredictable failure conditions. Make sure the plug from the power supply is oriented properly on the board. If you intend to program or erase flash memory, you should also get a regulated +12VDC supply.

CAUTION

The power-supply plug is keyed, so it should easily attach to the board; if you forced it on, you may have attached it backward. Powering up the board through a backward plug may damage board components.

4. Turn on the power.

2.5 DEFAULT EVALUATION BOARD JUMPER AND DIP-SWITCH SETTINGS

The evaluation board ships with the 8x930Ax microcontroller installed. The default jumper and DIP-switch settings correspond to configuration for code downloading and debugging through RISM in the binary nonpage mode. Table 2-1 and Table 2-2 provide the default settings. To use another mode (i.e., binary page, source page, or source nonpage), refer to Chapter 6, "Jumper and DIP-Switch Settings" for the specific jumper and DIP-switch settings. The 8x930 Family USB evaluation board's default jumper setting are listed in Table 2-1 below; and the default DIP-switch settings are listed in Table 2-2.

Table 2-1. Evaluation Board's Default Jumper Settings

Jumper	Setting	Jumper	Setting
JW1	open	JW13	short
JW2	2–3	JW14	1–2
JW3	2–3	JW15	1–2
JW4	1–2	JW16	2–4
JW6	1–2	JW17	short
JW7	1–2	JW18	short
JW8	2–3	JW19	2–3
JW9	2–3	JW20	1–2
JW10	1–2	JW21	1–2
JW11	1–2	JW22	2–3
JW12	2–3	JW23	short

NOTE

According to the *Universal Serial Bus Specification*, Rev. 1.0, full speed devices should have a pull up resistor on their D+ line and low speed devices should have a pull up resistor on their D- line. The design of the evaluation board accounts for these resistors and allows the choice of low-speed or full-speed configurations through the use of JW7. However, in order to allow some debugging flexibility, the board design allows the user to disable (turn off the power to) the pull up resistor through the use of jumper JW6 setting and the reset pulse.

When JW6 is at setting 1-2, the pull up resistor is always enabled. When the user selects this position, he/she **should not** connect the USB cable to the USB host until after downloading (through the RS232 port) the code to the board and starting it (by issuing the GO command from the debugger).

Developers who do not want to disconnect and connect the USB cable each time they download new code to the board, need to have JW6 in position 2-3. This will allow the Reset signal to control the pull up resistor. The reset can be

caused through the push-button on the board and/or initiated from the debugger environment through the serial port (RS232). With each time a reset occurs, it will cause the emulation of a disconnect-connect action on the USB line. The reset causes the pull up resistor to first be disabled then re-enabled. In this scenario, when the USB cable is connected and the enumeration process is repeated, the address given to the board by the USB host will (in all probability) change.

It is worth mentioning that the use of JW6 in the position 2-3 will be most beneficial to developers that are using ROM-emulators or their own EPROMs (containing their own code) and replacing the EPROM provided with the board.

For developers who are using the USB SST (single step transaction) tool, they should setup JW6 to the 1-2 position. They also need to have the board connected to the USB host before they download the code (through the RS232 port) and issuing the GO command.

For more information on connect-disconnect signaling, the reader is encouraged to consult chapter 7 of the *Universal Serial Bus Specification*, Rev. 1.0. In order to get a good idea of the role of JW6 and the reset, one should refer to the schematics provided in Appendix C, "Schematics."

Table 2-2. Evaluation Board's Default DIP-Switch Settings

Switch	Position
RD1	on
RD0	on
ADS2	on
ADS1	on
ADS0	off
ADT1	on

Switch	Position
ADT0	on
EA#	on
MOD1	on
MOD0	off
UARTC	on
RISM	on

As a rule, to reconfigure the mode settings, change only the DIP-switch settings. To use the board in one of the following configurations, refer to the appropriate section:

- To use the evaluation board in stand-alone mode with your own EPROM, refer to "Using the Board in Stand-Alone Mode" on page 2-9.
- To use the evaluation board with the pre-programmed 8x930Ax code, refer to "Jumper and DIP-Switch Settings for 8x930Ax Firmware" on page 2-10.

2.6 INTERFACING THE EVALUATION BOARD TO A DEBUGGER

The software tools used to develop code for the 8x930Ax microcontroller and to interface with this board offer a Windows*-based, integrated development environment. Configure the tool and the debugger environments (com port, baud rate, etc.) to create a functional link. Use the Windows graphical user interface to do this. However, read the following information before you begin configuration.

The microcontroller has a default start-up internal frequency of 3 MHz. You can change this frequency to 12 MHz by clearing the LC bit located in the PCON register. You can switch back to 3 MHz by resetting the LC bit. For details about this process, refer to the *8x930Ax, 8x930Hx Universal Serial Bus Microcontroller User's Manual*.

The RISM code for this board has been modified to increase the stability of the board-PC communication link during debugging. With previous products, the flexibility of frequency changes for power saving purposes sometimes adversely affected the baud rate and serial communication. The changes in RISM code should eliminate these problems. To establish communication between the evaluation board and the debugger (Keil, PLC, or Tasking software), use either the external-serial link or the internal-serial link. We recommend the external-serial link.

2.6.1 Using the External-Serial Port

It is strongly recommended to use the external-serial port to communicate with the debugger environment. When you use the external-serial port, the baud rate is **19200** (the debugger should be set at the same rate). With all internal timers free, you can use them for your specific application. The internal-serial port is also free; allowing it to be used for any specific applications. The UART, however, uses INT0; therefore, you cannot use this interrupt.

The external-serial port is labeled as UART. To enable this port, set the UARTC switch to the "on" position. When using this port, programmers can set or clear the LC (low clock) bit in their code by using the SETB and CLR instructions. There are no restrictions to the programming code.

2.6.2 Using the Internal-Serial Port

It is NOT recommended to use the internal-serial port to communicate with the debugger. However, if you use this serial port, the baud rate is fixed at **9600** (set the debugger for the same rate). RISM uses either timer one or timer two to generate the baud rate; therefore leaving the timer unavailable for your application. In addition, the internal-serial port is no longer free to use since it is dedicated to handling debug information.

Both external interrupts, INT0 and INT1, are free during this time. When using the internal-serial port, the programmer can set or clear the LC bit, however, in doing so, the programmer cannot use the SETB and CLR instructions. The programmer **must** use an extended call to a routine inside RISM that sets/clears the bit.

The programmer should use the following:

- "ECALL 0FF:0083h" to set the LC bit. Do not use "SETB LC".
- "ECALL 0FF:008Bh" to clear the LC bit. Do not use "CLR LC".

2.7 USING THE BOARD IN STAND-ALONE MODE

In stand-alone mode, the board is used without connections to the debugger for code downloading and debugging. Also in this mode, socket A is mapped to the entire upper-memory blocks, the external UART is disabled, and the mode DIP switches are overridden. Essentially, the upper memory becomes ROM only. If you use this mode, use a device programmer to program your executable application code on an EPROM or flash memory device. Then, use the device you just programmed to replace the EPROM that shipped with the board. The EPROM is in socket A (refer to Figure 2-1 on page 2-3).

NOTE

If you use the 32-Kbyte EPROM device, change the RISM DIP-switch setting to "off". Leave the remainder of the DIP switches and jumpers in their default positions.

When using the 32-Kbyte EPROM device, be sure to change the RISM DIP-switch setting to the "off" position. The remainder of the DIP switches and jumpers should be left in their default positions. Your code **must** start at 0FF:0000h. The RAM locations are mapped between 00:0000h and 01:FFFFh. For more information about the board's memory map, refer to Chapter 4, "Memory Configuration".

2.8 USING THE ON-BOARD 8x930Ax firmware

The EPROM that contains RISM and allows communication with the debugger environment also contains some 8x930Ax peripheral controller firmware. This firmware is provided to enable you to use the board in stand-alone mode (no debugger interface), as well as allowing you to use a USB cable to connect the board's upstream port to a USB host PC.

If you use the 8x930Ax code, the board acts as a function. When connecting the evaluation board to the USB host, the host will request that a driver be loaded—this means the enumeration process has taken place. If you use the 8x930Ax code exactly as it is programmed in the EPROM, you will have little flexibility. You will only be allowed to double check the connections and the USB host PC functionality, and see some USB transaction on the bus (with the help of a protocol analyzer).

The 8x930Ax firmware source code, which is programmed in the EPROM, is available from the Intel web page. You can download this code and tailor it to your application. The source code is located under software support at: *http://developer.intel.com/design/usb/swsup*. Refer to “World Wide Web (WWW)” on page 1-7 for additional web site locations.

2.8.1 Jumper and DIP-Switch Settings for 8x930Ax Firmware

To use the 8x930Ax firmware programmed in the EPROM, **change the JW9 jumper setting by connecting pins 1 and 2**. Leave the remainder of the jumper and DIP switches in their default positions.

3

Hardware and Functional Overview

CHAPTER 3
HARDWARE AND FUNCTIONAL OVERVIEW

This chapter describes functional units of the 8x930 Family USB evaluation board. It includes a block diagram of the board, a diagram of the board's major components, and brief descriptions of each functional section. Appendix C includes schematics of the evaluation board.

3.1 BLOCK AND COMPONENT DIAGRAMS OF THE BOARD

Figure 3-1 is a block diagram of the evaluation board. The diagram illustrates the primary board components. As shipped, the board has a 128-Kbyte SRAM and a 32-Kbyte EPROM. It also comes with the 8x930Ax microcontroller installed. You can replace the 8x930Ax microcontroller with a adapter board for added functionality.

WARNING

Adapter boards can only be used with he four port evaluation board.

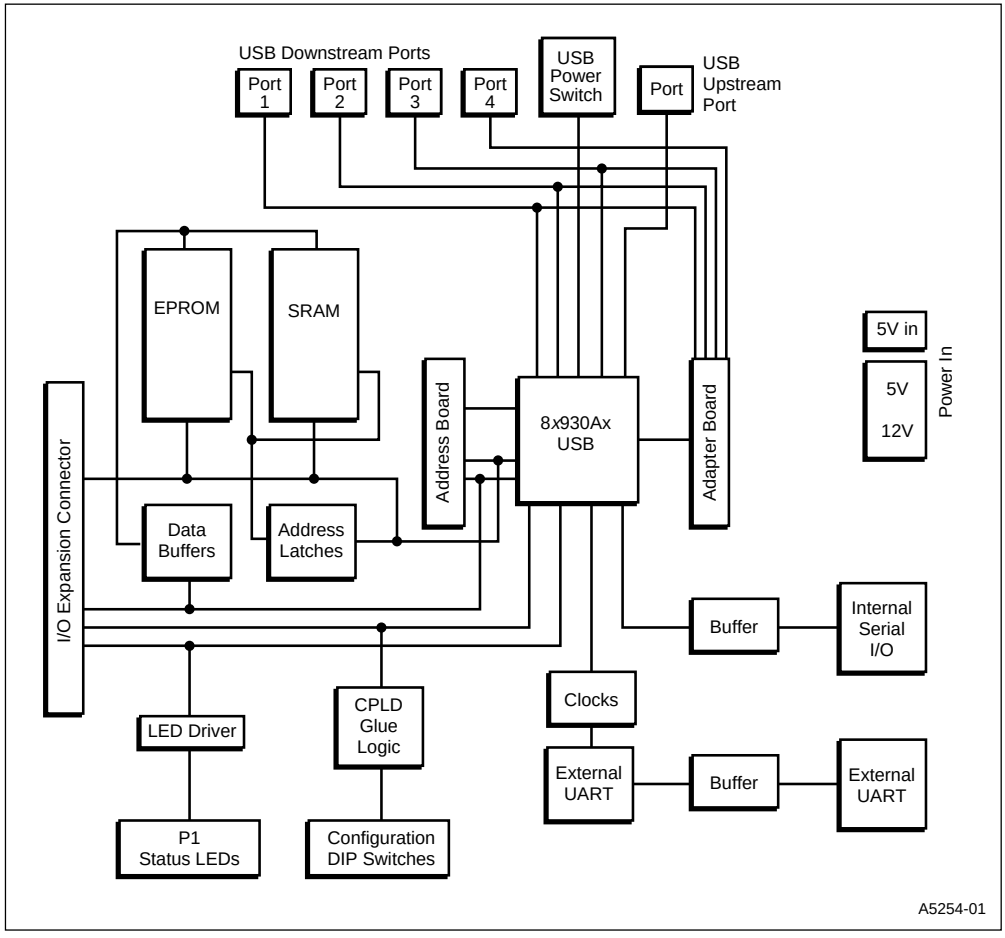


Figure 3-1. 8x930 Family USB Evaluation Board Block Diagram

Figure 3-2 on page 3-3 is a component-level diagram of the evaluation board. The evaluation board is a four-layer, printed circuit board with separate power and ground planes. It operates at 12 MHz and will consume a maximum of 500 mW. Component details are discussed further in this chapter.

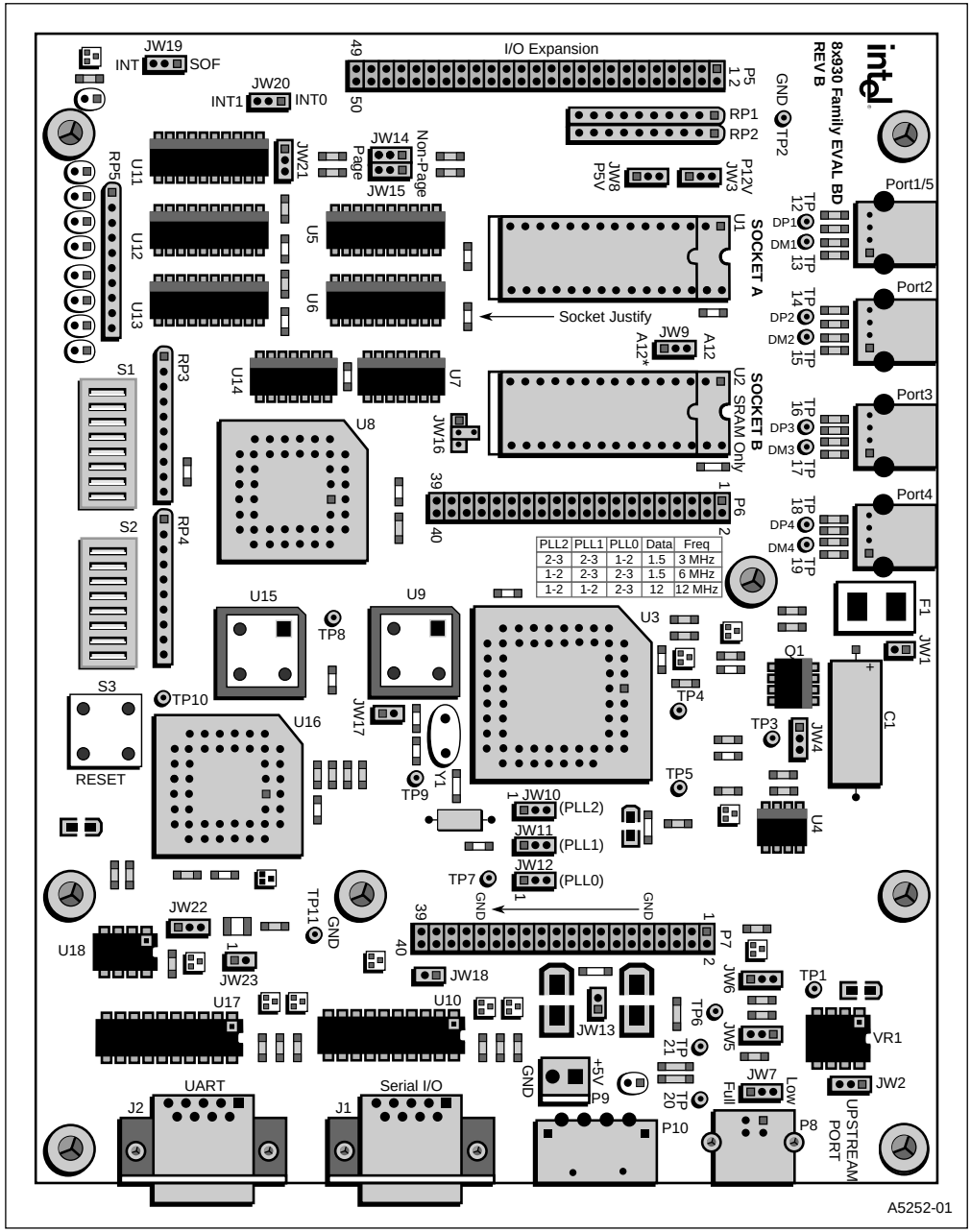


Figure 3-2. Component-level Diagram of the 8x930 USB Evaluation Board

3.2 8X930AX MICROCONTROLLER FEATURES

The 8x930Ax microcontroller contains all the features of the MCS® 251 architecture and provides a USB interface for a personal computer (PC) peripheral. This microcontroller supports the four types of USB data transfers: control, isochronous, interrupt, and bulk. You can select the number of function endpoint pairs (four or six) and whether the USB reset is separate from chip reset. Each endpoint pair has a transmit first-in, first-out (FIFO) and a receive FIFO data buffer. If you would like to know more about the microcontroller than is discussed in this section, refer to the *8x930Ax, 8x930Hx Universal Serial Bus Microcontroller User's Manual*.

3.3 8X930XX AND 8X931XX MICROCONTROLLER ADAPTER BOARDS

The 8x930xx microcontroller adapter boards contain all the features of the MCS® 251 architecture and provides a USB interface for a personal computer (PC) peripheral. This microcontroller supports the four types of USB data transfers: control, isochronous, interrupt, and bulk. Each endpoint pair has a transmit first-in, first-out (FIFO) and a receive FIFO data buffer.

The 8x931xx microcontroller adapter boards contain all the features of the MCS 51 architecture and provides a USB interface for a personal computer (PC) peripheral. This microcontroller supports the four types of USB data transfers: control, isochronous, interrupt, and bulk. Each endpoint pair has a transmit first-in, first-out (FIFO) and a receive FIFO data buffer.

Both families of adapter boards will offer extended capabilities to the 8x930Ax microcontroller shipped with the kit. Such added capabilities are: the evaluation board acting as both a hub and a function, increasing the number of downstream ports; and a lower cost solution by using the 8x931 family microcontrollers. Refer to the specific documentation for information on each adapter board and its functionality.

3.4 HOST INTERFACE

The host interface is a connection between the host-PC serial port (com1 or com2) and the 8x930 serial I/O (SIO) on J1 or the UART device on J2. The UARTC DIP switch selects between these two possible serial interfaces. J1 and J2 are standard RS-232, 9-pin, female connectors.

3.5 INTERFACES

All the address/data bus pins; port pins; and other necessary device pins, including reset, power and GND, are available through a 50-pin header on the side of the board (see Table 3-1).Use either a ribbon cable or a female connector on the application board to connect the application board to the 50-pin header connection (see Figure 3-2 on page 3-3).

NOTE

The USB signals are connected to USB-standard connectors and do **not** go to the 50-pin header.

Table 3-1. 50-Pin Header Signal List

Pin Name	Pin Number	Pin Number	Pin Name
V_{CC}	1	2	V_{CC}
A8	3	4	AD0
A9	5	6	AD1
A10	7	8	AD2
A11	9	10	AD3
A12	11	12	AD4
A13	13	14	AD5
A14	15	16	AD6
A15	17	18	AD7
GND	19	20	GND
GND	21	22	GND
P1.0	23	24	ALE
P1.1	25	26	PSEN#
P1.2	27	28	P3.6
P1.3	29	30	P3.7
P1.4	31	32	Memdisabl#
P1.5	33	34	NC
P1.6	35	36	RESET#
P1.7	37	38	NC
V_{CC}	39	40	V_{CC}
NC	41	42	SOF#
P3.4	43	44	P3.0
P3.5	45	46	P3.1
P3.6	47	48	P3.2
P3.7	49	50	P3.3

3.6 USB PORTS

The evaluation board has one upstream port and four downstream ports (see Figure 3-2 on page 3-3).

- **Upstream port**: The upstream port has a configureable pull-up resistor. It allows you to select full- or low-speed enumeration. The port also contains 33-Ω series termination resistors for the data lines. This port does not have EMI filtering.
- **Downstream ports**: The downstream ports contain 15 KΩ pull-down resistors and 33-Ω series termination resistors. Three of these ports connect to the 68-pin socket for the device; the fourth port connects only to the adapter board header.

3.7 LEDS

A bank of eight LEDs provides visual indication of the state of Port 1 pins. You can write different values to Port 1 in various stages of the user code and use the LEDs to verify correct program behavior. Although the eighth LED is connected to P1.7, it may not show the port register value. If this occurs, P1.7 can be configured as A17.

3.8 RESET BUTTON

The evaluation board includes a reset button used to manually reset the CPU. The inverted reset signal is available via the 50-pin header.

3.9 SERIAL PORTS

The evaluation board contains two, separate RS-232 serial ports. One is driven from the internal SIO port to the 8x930 Family device. The other is driven by an external UART device, the 16C550. Normally, the external UART interfaces with a PC through connector J2; so, RISM can use the external UART and leave the internal SIO for the application program to use.

The 16C550 is memory mapped to locations FF:FFC0h–FF:FFDFh.

3.10 RS-232 CONNECTORS

The evaluation board has two DB-9S female connectors and uses a standard RS-232 cable to communicate with the evaluation board and host PC. The cable is included as part of the development kit.

3.11 POWER SUPPLY CONNECTORS

The evaluation board contains two power connectors: P10 and P9 (see Figure 3-2). When you power up the board using either connector, the following will occur:

- LED CR1 glows. (The green LED near the power supply connectors)
- LED CR3 (the RUN LED) glows when the 8x930 bus is active. (The green LED near the 50 pin header.)

WARNING

You **must** use a +5 volt-regulated power supply. Lower voltage may not operate the evaluation board. Higher voltage may damage the board. An unregulated power supply may cause unpredictable failure conditions. Make sure the plug from the power supply is oriented properly on the board. If you intend to program or erase Flash memory, you should also use a regulated +12VDC supply.

3.11.1 Power Connector P10

P10 is compatible with a standard-PC, power supply connector. It has the following attributes: +12 volt on Pin 1, GND on pins 2 and 3, and +5 volt on pin 4. +12 volt is required only for Flash programming. When powered, the LEDs glow as indicated previously.

3.11.2 Power Connector P9

Use P9 to power up the board if you are **not** going to program a Flash chip. Connect +5 volt to Pin 1 and GND to Pin 2. When powered, the LEDs glow as indicated previously.

3.12 USER RESET VECTOR

After every reset, the CPU starts executing from external memory page address FF:0000h. This is where the RISM resides; it takes up approximately 3 Kbytes. RISM assumes application code starts at location 00:4000h. RISM establishes communication with the host PC, sets up the scratch pad and variables, and downloads the user code. Upon receiving the GO command, RISM loads the program counter with address 4000h; the CPU starts executing from this address.

3.13 USER INTERRUPT VECTOR TABLE

Any application program that resides at location 4000h **must** have an interrupt vector table just as if it started at location 0—except, 4000h must be added to every vector address. For example, external interrupt 0 will be at address 4003h followed by the rest of the table 400Bh, 4013h, etc. The vector table should follow the pattern defined in the *8x930Ax, 8x930Hx Universal Serial Bus Microcontroller User's Manual*. Refer to "Related Documents" on page 1-5 for ordering instructions.

4

Memory Configuration

CHAPTER 4
MEMORY CONFIGURATION

The 8*x*930 evaluation board contains two 32-pin DIP-switch sockets (sockets A and B) for memory devices. Although any type of memory can be plugged into socket A, this socket is designed to be addressed as code memory. Socket B is designed to be addressed as data memory; it accepts SRAM only. As shipped, the sockets are populated as follows:

- Socket A is populated with a 32-Kbyte EPROM containing RISM (Reduced Instruction Set Monitor) and applications programs.
- Socket B is populated with a 128-Kbyte SRAM. The CPLD device (EPM7032) performs selection, timing, and control, based on the DIP-switch settings.

4.1 SOCKET A (EPROM OR FLASH)

The evaluation board is shipped with RISM installed in an EPROM within socket A. With the RISM DIP switch in the "on" position (i.e., running RISM), RISM within this EPROM is mapped to FF:0000h–FF:1000h. This RISM mapping results from the CPLD device decoding the DIP-switch settings for source/binary, page/nonpage, and RISM "on" operation.

When the RISM DIP switch is in the "off" position, the CPLD device maps socket A from FF:E000h–FF:FFFFh. This is the stand-alone mode referenced in "Using the Board in Stand-Alone Mode" on page 2-9.

4.2 SOCKET B (SRAM) RISM DIP SWITCH SETTING

The following section discusses the RISM DIP-switch setting.

4.2.1 RISM DIP Switch Set to “ON”

If the RISM DIP switch is “on”, the SRAM is mapped into addresses FF:2000h–FF:DFFFh and 00:0000h–01:FFFFh (see Figure 4-1 on page 4-3). As a default, RISM assumes that the applications program code starts at 00:4000h.

NOTE

FF:2000 and 01:2000 address the same RAM byte. Since only 18 lines are available, only 4x64 Kbytes (a total of 256 Kbytes) can be addressed. So, the pages listed in the following table are all that exist as binary. The table also shows their hexadecimal equivalents:

Binary	—	Hexadecimal
XXXX XX00	Same as	00:
XXXX XX01	Same as	01:
XXXX XX10	Same as	FE:
XXXX XX11	Same as	FF:

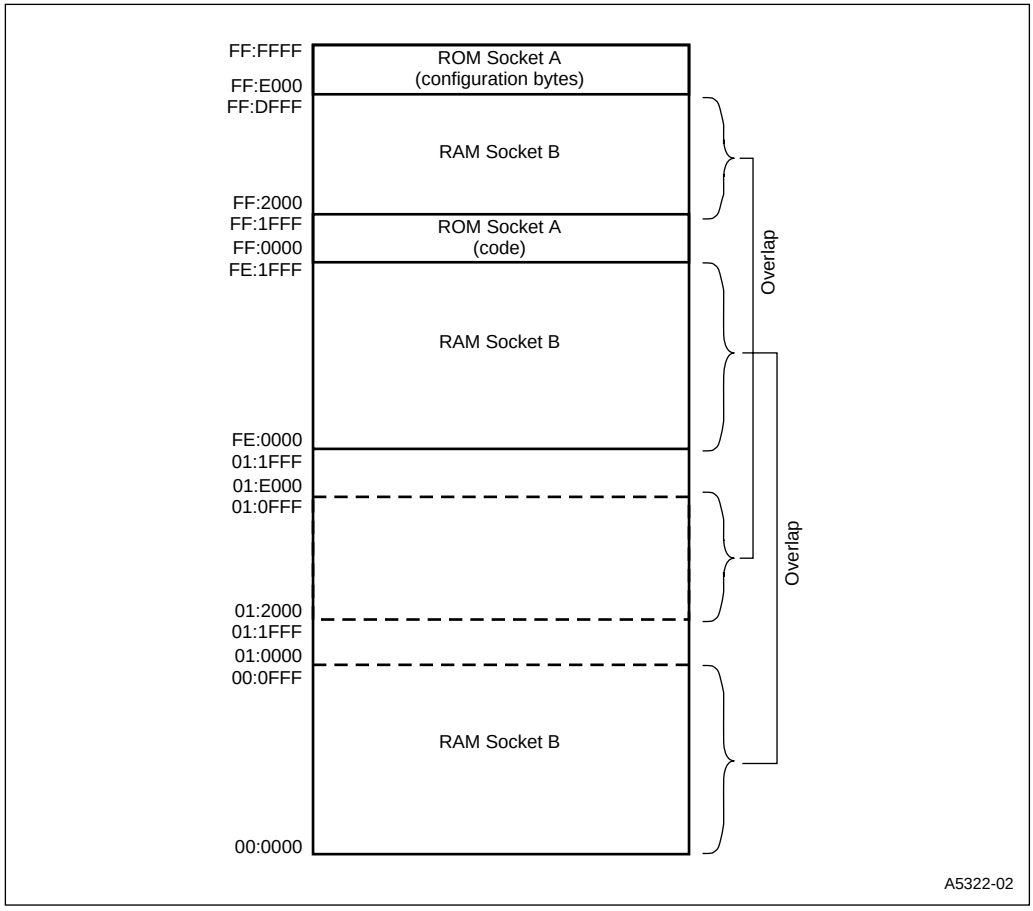


Figure 4-1. RISM = “ON” Memory Map

4.2.2 RISM DIP Switch Set to “OFF”

If the RISM DIP switch is “off” (i.e., not running RISM), the SRAM is mapped to addresses 00:0000h–01:FFFFh (see Figure 4-2).

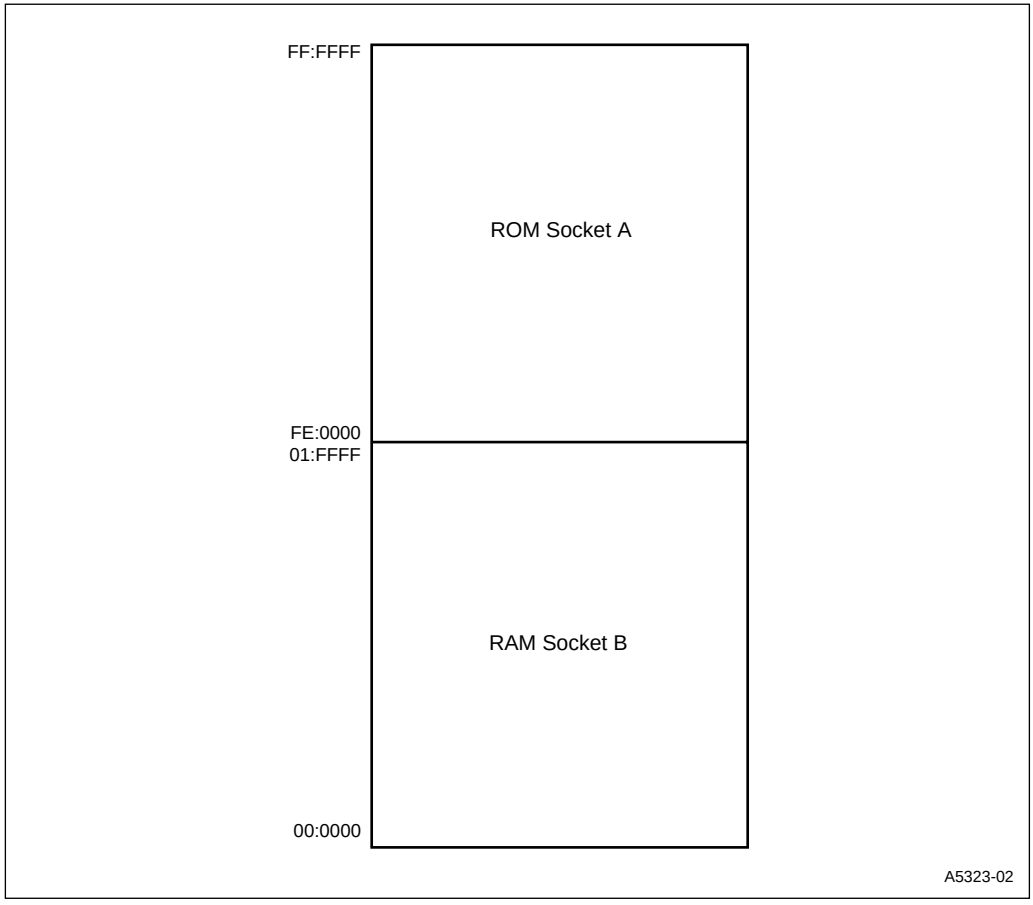


Figure 4-2. RISM = “OFF” Memory Map

4.3 CONFIGURATION BYTES

This section briefly discusses configuration bytes. For more details, refer to Chapter 16 “External Memory Interface” in the *8x930Ax, 8x930Hx Universal Serial Bus Microcontroller User’s Manual*.

The evaluation board’s configuration is established by the reset routine based on information stored in the configuration bytes. The evaluation board fetches configuration information from two user configuration bytes (UCONFIG0 and UCONFIG1); these bytes are located in code memory. Devices with no on-chip code memory fetch configuration data from external memory. Factory- programmed ROM devices use customer-provided configuration data supplied on floppy disk.

After each reset, the evaluation board reads two bytes from the top of region FF: to obtain configuration information. The top eight bytes in region FF are reserved; but, only the first two bytes are used for configuration information. For example, if you use a 32-Kbyte memory chip, the configuration bytes **must** reside at bytes FF:7FF8h and FF:7FF9h.

It should be noted that the configuration bytes should normally reside at the top of the EPROM device being used. For example, if a 32 Kbyte EPROM is used, the configuration bytes can be located at 7FF8h and 7FF9h within the device. The configuration bytes are mapped into memory at locations FF:FFF8h and FF:FFF9h respectively. Another example is if a 64 Kbyte EPROM is used; the configuration bytes would be located at FFF8h and FFF9h within the device, however, mapped into memory at FF:FFF8h and FF: FFF9h respectively. This is true for the default settings of binary non-paged mode and stand-alone mode. To use a different mode other than binary non-paged, you need to locate your configuration bytes at a different location. Refer to Chapter , “CPLD Equations”, page B-3 for more information about the location of code and configuration bytes in the EPROM.

RISM provides the chip configuration bytes for the evaluation board based on the setting of the DIP switches, MOD1 and MOD2. When using the board in the stand-alone mode (RISM DIP switch is set to “off”), write the configuration bytes in the appropriate locations. The bytes **must** be consistent with the DIP-switch settings.

5

RISM

CHAPTER 5
RISM

The Reduced Instruction Set Monitor (RISM) consists of a program located in the internal ROM of the microcontroller or other ROM on an evaluation board. This program provides primitive operations such as read memory, write memory, start execution, and stop execution. The communication software running on the host computer uses RISM commands to provide a complete user interface to the target board. This communication software is called ROM-monitor. It is part of a debugger environment.

NOTE

This chapter is written primarily for ROM-monitor developers.

The ROM-monitor communicates with the microcontroller by sending special RISM commands across the serial port from a personal computer (PC) at a fixed baud rate. Figure 5-1 shows the relationship between the evaluation board and a PC.

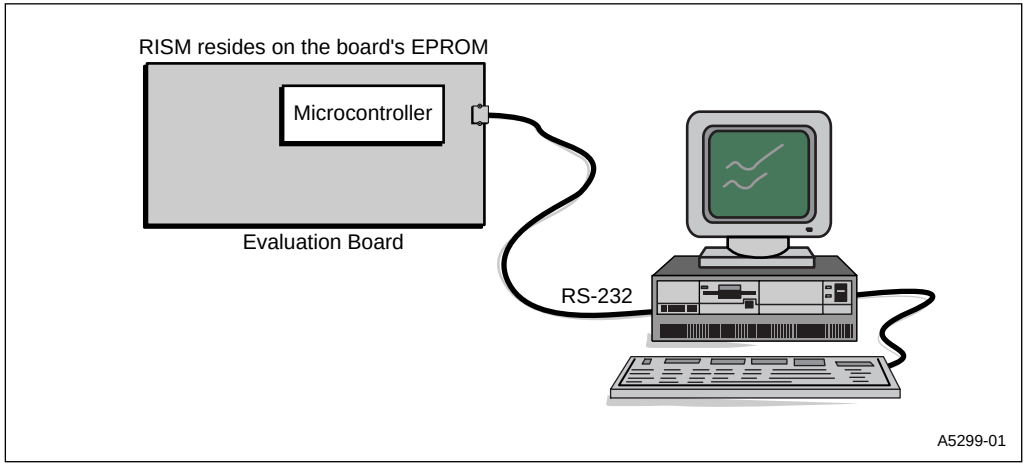


Figure 5-1. Serial Link Between the 8x930 Family USB Evaluation Board and a PC

NOTE

The structure and functionality of RISM described in this document is related to the MCS 251 microcontrollers and 8x930 Family USB devices. A similar implementation is used for the MCS 51 microcontrollers.

5.1 RISM STRUCTURE AND FUNCTIONALITY

Upon reset, RISM runs an initialization section of code. Then, it waits for commands from the host PC. The microcontroller enters a waiting loop called “Monitor_Pause” in which it waits to receive data characters across the serial port. A received character can be either data or a RISM command. Each command is one byte in length and has a value between 00H and 1FH. If you send data values less than 1FH, you **must** precede them with a Data Latch Enable (DLE) command to alert RISM that the next character should be stored in the DATA register.

When a receive interrupt occurs, RISM checks to see if the DLE flag is set or if the data received is greater than 1FH. If either of these conditions is true, RISM assumes that the received byte is data and reads it into a 32-bit First-In-Last-Out (FILO). Each time new data comes in, the DATA register shifts left by eight bits. If RISM identifies the received data as a command, RISM executes it. After completing the command, the device re-enters “Monitor_Pause”. Figure 5-2 shows the overall flow of the RISM code.

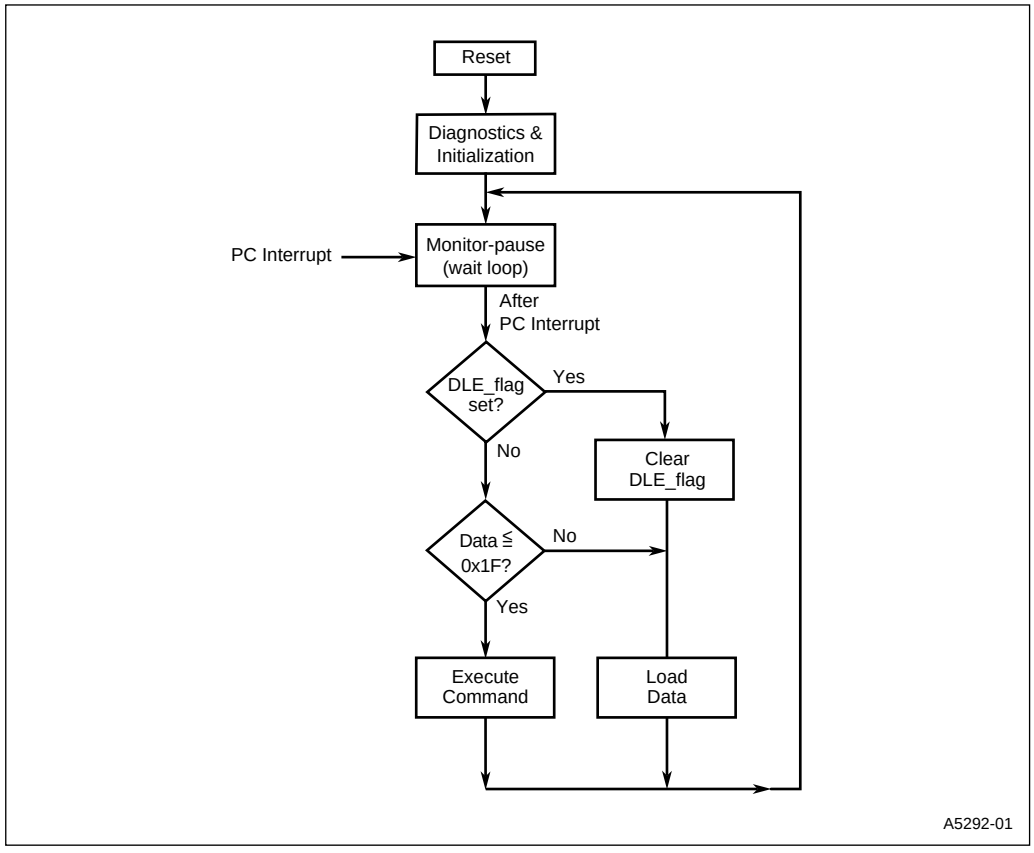


Figure 5-2. Flow of RISM Code

5.2 RISM RESOURCES

WARNING

Because the evaluation board and RISM use some of the same resources to communicate with the host PC, these resources may not be available to you. The restrictions on the availability of these resources depends on the board's application. Some restrictions may not apply if you are using the board in a stand-alone mode.

The affected resources are:

- RISM reserves RAM locations 29h–3Fh for RISM data registers. If the application program attempts to use these locations, unpredictable operation can result. RISM uses approximately 10 bytes of the stack.
- Memory locations 3E0h to 3FFh are reserved for RISM stack.
- RISM can handle communication to the debugger through the external UART or the internal SIO.
  - — The external UART uses either interrupt INT0 or INT1. The jumper settings determine which interrupt RISM uses. So, the interrupt that RISM uses is unavailable for application programs. The USB evaluation board's default interrupt setting is INT0.
  - — If the internal SIO is used, then one of the timers (1 OR 2) is also not available to the user. The USB evaluation board's default timer setting is 2.
- RISM has a release number byte at memory location FF:00FFh.

5.3 GENERAL RISM INFORMATION

Four separate RISMs reside at region FF: in a non-volatile memory device of the 32-Kbyte EPROM. Region FF: also contains the configuration bytes to set up the modes for the 8x930 Family USB evaluation board.

The RISMs are:

- Binary page
- Binary nonpage
- Source page
- Source nonpage

When the evaluation board arrives, it is set for binary, nonpage RISM. Use the board with this setting, or reset it for one of the other choices. If you choose another RISM, reset the MOD1 and MOD2 switches in the DIP-switch bank. The mode DIP-switch settings **must** correspond to the configuration bytes.

The host PC debugger sends its commands to RISM, and the 8x930 Family USB evaluation board executes them. RISM downloads user object code, one byte at a time, into the SRAM starting at location 00:4000h. RISM also sets the user's program counter to the beginning of the user code when it receives either a GO or RUN command from the debugger.

RISM source code is available to the public at no cost and with no technical support. The code is available via the following World Wide Web site:

http://www.intel.com/design/usb/swsup/

5.4 RISM STACK POINTER AREA

RISM sets the stack pointer to location 3E0h. You can initialize the stack pointer to a different area; but, you should leave about 10 bytes for RISM to use for debugging. RISM saves and restores the user stack during the execution of most debugger commands.

5.5 RISM CONFIGURATION

NOTE

This section is important **only** if you want to reset the RISM.

The RISM code in this USB evaluation board has been configured to meet the 8x930 USB microcontroller architecture, but you can reconfigure it for MCS 251 boards. The code is available via the following World Wide Web site:

http://www.intel.com/design/usb/swsup/

The board ships with the following RISM-code configuration:

- **USBRISM**: Configured for the USB board specifications (equ 1).
- **SRCMODE**: Configured for either source-mode or binary-mode (equ 1 and equ 0).
- **EXT_MEMORY**: Configured as 00:4000H (004000H).
- **EXTINTR**: Configures the external UART so it can use INT0 (equ 0).
- **TIMER**: Configures the baud rate generator for timer 2 (equ 2).

5.6 RISM REGISTERS AND BITS

RISM defines eight registers (in internal RAM locations) that are used throughout the code to allow navigation through the RISM structure, data storage, and context switching.

The registers are:

- mod
- selector
- RISM_DATA
- RISM_ADDR
- user_pc
- char
- Baud_rate
- Baud_rate1

The registers mod and selector are used in the form of bits. Each bit serves a special purpose in running the RISM code and communicating with it. Review the RISM code to learn more about these registers and bits.

5.7 INTERRUPT ENABLE AND PRIORITY REGISTERS

Do not clear the global enable bit or the serial-port interrupt enable bit. Clearing either bit terminates communication between the evaluation board and the host PC. Change the contents of the interrupt enable registers by using logical OR or AND operations. For more details about interrupts, refer to the *8x930Ax, 8x930Hx Universal Serial Bus User's Manual*.

5.8 INTERRUPT VECTORS OFF SET

All available application interrupt vectors are re-directed to:

EXT_MEMORY + vector_offset

(Where *EXT_MEMORY* = 00:4000H on the evaluation board.)

(Where *vector_offset* is the real ISR Vector Address minus FF:0000H. For example, 00:4033H is the redirected address for PCA interrupts on evaluation board.)

5.9 RISM COMMANDS

RISM has 32 possible commands; each command has an associated code. The code is a hexadecimal number between 00 and 1F inclusive. In addition, each command has a special function represented by a set of instructions in RISM. Table 5-1 shows the set of RISM commands and explains each command's task(s).

Table 5-1. RISM Commands

Value	Name	Code	Description
1	Set_DLE	00H	Sets the DLE flag. This forces the next byte received by the RISM to be treated as data even if its value corresponds to a RISM command. The DLE flag is cleared as soon as the byte is read.
2	Xmit	01H	Transmits the lower-eight bits of the 32-bit DATA register to the host, and shifts the DATA register eight places to the right.
3	Xmita	02H	Transmits the lower-eight bits of the 32-bit DATA register to the host, shifts the DATA register eight places to the right, and increments the ADDR register by one.
4	RESERVED	03H	Reserved
5	Read_byte	04H	Reads the byte of memory pointed to by the ADDR register and places the result in the least significant byte of the DATA register.
6	Read_word	05H	Reads the word of memory pointed to by the ADDR register and places the result in the least significant word of the DATA register. SFR's **cannot** be read from as words.

Table 5-1. RISM Commands (Continued)

Value	Name	Code	Description
7	Read_long	06H	Reads the double-word of memory pointed to by the address register and places the result in the 32-bit DATA register. SFR's **cannot** be read from as dwords.
8	Write_byte	07H	Stores the least significant byte of the DATA register in the byte of memory pointed to by the ADDR register and increments the ADDR register by one, so it points at the next memory byte.
9	Write_word	08H	Stores the least significant word of the DATA register in the word of memory pointed to by the ADDR register and increments the ADDR register by two, so it points to the next memory word. SFR's **cannot** be written to as words.
10	Write_long	09H	Stores the 32-bit DATA register in the double-word of memory pointed to by the ADDR register and increments the ADDR register by four, so it points at the next memory double-word. SFR's **cannot** be written to as dwords.
11	Data_to_addr	0AH	Loads the 32-bit ADDR register with the 32-bit DATA register. REGISTER and SFR addresses have byte (8-bit) address. Byte registers have addresses 0x00 to 0x0F; and word registers have addresses 0x00 to 0x1E. Address 0x38 (DR56) is the only addressable dword register address now. SFRs have addresses 0x80 to 0xFF. Other memory locations should use 32-bit addresses.
12	Indirect0	0BH	Reads the memory dword pointed to by the ADDR and stores it into the ADDR register.
13	Read_PSW	0CH	Loads the low word of the 32-bit DATA register with the PSW and PSW1 associated with the user's code. PSW loads at the lowest byte of the DATA register.
14	Write_PSW	0DH	Loads the PSW and PSW1 associated with the user's code from the low word of the DATA register. PSW loads at the lowest byte of the DATA register. PSW1 is updated last, so shared bits of PSW and PSW1 are dictated by PSW1.
15	Read_SP	0EH	Loads the DATA register with the SP (Stack Pointer) associated with the user's code.
16	Write_SP	0FH	Loads the SP (Stack Pointer) associated with the user's code from the DATA register.
17	Read_PC	10H	Loads the 32-bit DATA register with the 32-bit user_pc (Program Counter) RISM register associated with the user's code.

Table 5-1. RISM Commands (Continued)

Value	Name	Code	Description
18	Write_PC	11H	Loads the 32-bit user_pc (Program Counter) RISM register associated with the user's code from the 32-bit DATA register.
19	GO	12H	Starts the execution of user code at the address in the user_pc register. As long as the user program does not disable interrupts, the monitor will run in the background servicing serial-port interrupts.
20	Halt	13H	Stops the execution of user code and returns to the RISM "Monitor_Pause" loop. The user program Program Counter restores in the user_pc (Program Counter) RISM register.
21	Report	14H	Loads the lowest byte of the DATA register with status information and transmits the value back to the host. (Status values are 00=stopped, 01=running, 02=trapped[at break].)
22	RESERVED	15H	Reserved
23	RESERVED	16H	Reserved
24	Read_baud	17H	Reads Baud_rate1 and Baud_rate into the low word of the DATA register. • For external UART, DATA+3 contains the low byte of the Decimal Divisor whereas DATA+2 contains the high byte of the Decimal Divisor. • For TIMER 1 as baud generator, DATA+3 contains the reload value and DATA+2 contains the SMOD1 value (0 or 1). • For TIMER 2, DATA+3 and DATA+2 contain the RCAP2L and RCAP2H value respectively.
25	Write_selector	18H	Loads the SELECTOR byte with the lowest byte of the DATA register. The valid selector register values are 01= for REGs, 08=for SFRs, and 00= for other locations. **Execute this command before commands 5, 6, 7, 8, 9, and 10.**
26	RESERVED	19H	Reserved
27	RESERVED	1AH	Reserved
28	Step	1BH	Single-steps one instruction of user code and returns to RISM "Monitor_Pause". The user code Program Counter for next instruction is restored in the RISM register user_pc.

Table 5-1. RISM Commands (Continued)

Value	Name	Code	Description
29	Write_baud	1CH	Writes the low word of the DATA register to Baud_rate and Baud_rate1 and sets the baud rate. It assumes the value of the baud rate is already in the DATA register. • For external UART, DATA+3 contains the low byte of the Decimal Divisor whereas DATA+2 contains the high byte of the Decimal Divisor. • For TIMER 1 as baud generator, DATA+3 contains the reload value and DATA+2 contains the SMOD1 value (0 or 1). • For Timer 2, DATA+3 and DATA+2 contain the RCAP2L and RCAP2H values respectively. (Check READ_BAUD).
30	Read_frequency	1DH	Reads the value of the frequency from SYS_FREQ to the DATA register. The value is read to the DATA register in BCD format representing the frequency in Hz.
31	Report_device	1EH	Loads the low byte of the DATA register (data-low) with DEVICE_ID.
32	RESERVED	1F	Reserved

5.10 EXAMPLES OF RISM COMMAND USE

Example 1: The goal is to download four bytes of data from the PC to the target board as a long-word in the DATA register. The four bytes are: 0x13, 0x22, 0x00, and 0x33.

Procedure: Send the following sequence of characters over the serial port.

1. Send 0x00.

   This character corresponds to the Set-DLE command. It tells RISM that the next character should be treated as data.

2. Send 0x13

   This character is less than 0x1F. It was sent after the DLE flag was set. It is treated as data and stored in DATA+3.

3. Send 0x22

   This character is greater than 0x1F. It is stored in DATA+3. The previous character is shifted to DATA+2.

4. Send 0x00

   This character is sent to set the DLE flag so that the next character will be treated as data.

5. Send 0x00

   This character is less than 0x1F. It was sent after the DLE flag was set. It is treated as data and stored in DATA+3. The other data characters are shifted left.

6. Send 0x33

   This character is greater than 0x1F. It is treated as data and stored in DATA+3. The other data characters are shifted left.

   At this point, we have:

   a. Content (DATA) = 0x13(Highest byte)
   b. Content (DATA+1) = 0x22
   c. Content (DATA+2) = 0x00
   d. Content (DATA+3) = 0x33(Lowest byte)

Example 2: The goal is to write the 32-bit data 0x00FF4000 to the ADDRESS register.

Procedure: Complete the following steps.

1. Send 0x00

   This character corresponds to the Set-DLE command. It tells the RISM that the next character should be treated as data.

2. Send 0x00

   This character is less than 0x1F; it was sent after the DLE flag was set. It is treated as data and stored in DATA+3.

3. Send 0xFF

   This character is greater than 0x1F; it is stored in DATA+3. The previous character shifts to DATA+2.

4. Send 0x40

   This character is greater than 0x1F. It is treated as data and stored in DATA+3. The other data characters shift left.

5. Send 0x00

   This character is sent to set the DLE flag so that the next character is treated as data.

6. Send 0x00

   This character is less than 0x1F; and it was sent after the DLE flag was set. It is treated as data and is stored in DATA+3. The other data characters shift left.

   At this point, we have:

   a. Content (DATA) = 0x00(Highest byte)
   b. Content (DATA+1) = 0xFF
   c. Content (DATA+2) = 0x40
   d. Content (DATA+3) = 0x00(Lowest byte)

7. Send 0x0A

   This character corresponds to the DATA_TO_ADDRESS command. After RISM receives this command, it copies the dword 0x00FF4000 from the DATA register to the ADDR register.

   At this point, we have:

   a. Content (ADDR) = 0x00 (Highest byte)
   b. Content (ADDR+1) = 0xFF
   c. Content (ADDR+2) = 0x40
   d. Content (ADDR+3) = 0x00 (Lowest byte)

5.11 IMPLEMENTING STEPS AND BREAKPOINTS

To step through user code, use the Step command provided by RISM. As mentioned above, this command will single-step one instruction of user code and return to RISM “Monitor_Pause”. The user code Program Counter for the next instruction is restored in the RISM register user_pc.

To implement a breakpoint at a specific location in user code, take advantage of the Trap instruction. This breakpoint implementation supposes that the Trap has the highest interrupt priority level. It also requires other interrupts, except the serial interrupt, to be disabled so they stay in the breakpoint location—even if an interrupt occurs.

The following procedure describes a breakpoint implementation:

1. The debugger software **must** replace the user's instruction with a Trap instruction.

   Instructions are encoded with different lengths: some instructions require one byte, while others require two or more bytes. When the Trap instruction has to replace a user instruction, the number of encoding bytes may not match; so, ROM-monitor developers **must** use extra caution when replacing instructions. These developers may need to shift user code or use other methods.

2. The debugger must store the user's instruction in a buffer until the user disables the breakpoint.

3. When the Trap is encountered as part of user code (instead of the user's instruction), the Trap interrupt occurs. RISM takes over using the Trap ISR and the break service routine. The break service routine performs context switching and routes code execution to the RISM wait loop: "Monitor_Pause".

4. To disable the breakpoint, the debugger should reinstall the user's instruction and delete the Trap instruction.

5.12 USB RESET SEQUENCE

This subsection explains some of the steps performed in RISM after a reset occurs. It discusses only the points relevant to the interface from a debugger's point of view (Figure 5-3 on page 5-14 also shows the flow):

1. A reset occurs.
2. RISM disables the serial interrupt.
3. RISM checks the power bit to determine if the reset was a warm or cold reset.

- If the power bit **does not** equal 1 **and** the mod.7 bit **does not** equal 0, RISM continues at the warm reset label. RISM then completes the following procedure:
  a. RISM sends the value 01h to the PC debugger to indicate that RISM is running.
  b. RISM sends this value because it assumes the debugger (ROM-monitor) has asked for a current status (using the Report command), but RISM cannot get the signal because of the reset time and because the serial interrupt is disabled. Voluntarily sending the running status can help the debugger maintain contact with the board. It may not be necessary to send the status if the debugger can try on multiple occasions to determine the status —without loosing contact with the board.
  c. RISM enables the serial interrupt.
  d. RISM switches contexts and allows the user code to start at the beginning **regardless** of where it was when you entered the reset.
- If either of the following is true, RISM continues at the cold reset label: 1) the power bit **does** equal 1; **or**, 2) the power bit **does not** equal 1, but mod.7 equals 0. RISM code then complete the following procedure:
  a. RISM clears the power bit.
  b. The code execution stays in RISM. To provide a visual indication that the board and RISM are working, RISM displays a repeatable pattern in the Port 1 LEDs.
  c. RISM goes to the RISM wait loop.

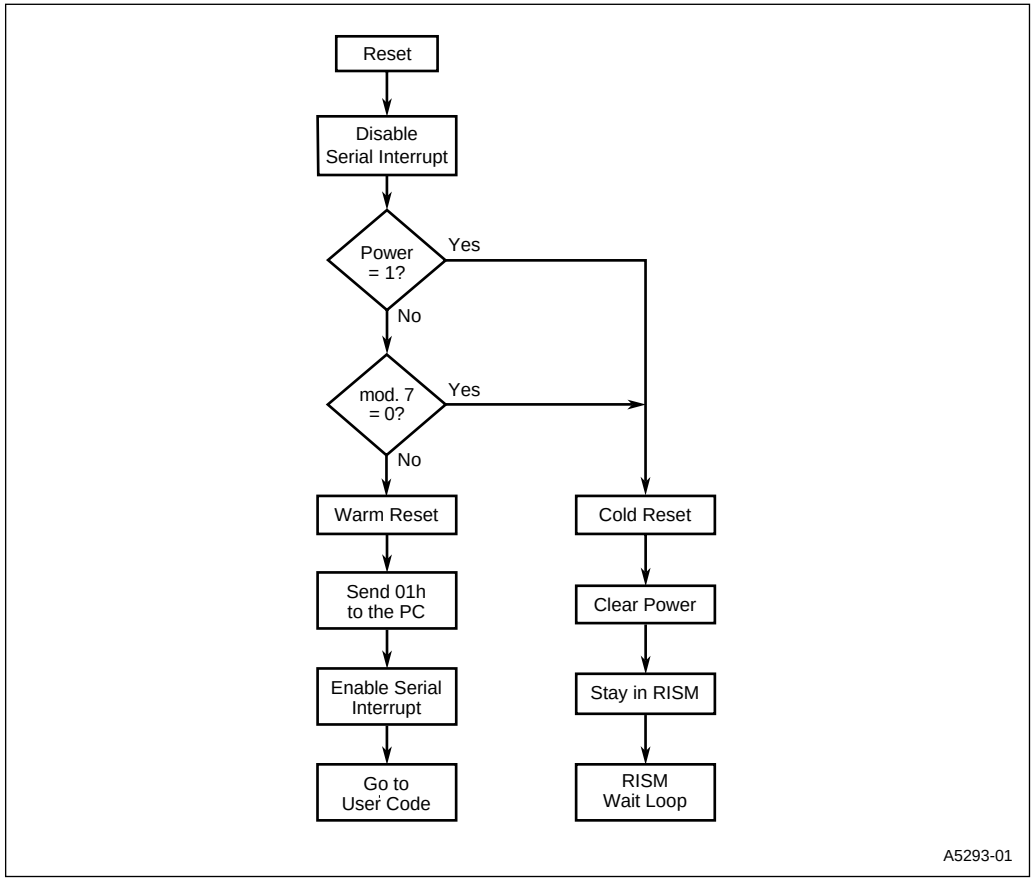


Figure 5-3. USB Reset Sequence

6

Jumper and DIP-Switch Settings

CHAPTER 6
JUMPER AND DIP-SWITCH SETTINGS

Jumpers and DIP switches provide a high degree of flexibility and functionality to the evaluation board. They also provide a variety of benefits including: variable memory size, memory type, page versus nonpage modes, binary versus source, and UART versus SIO selections.

This chapter describes the jumpers and DIP switches on the evaluation board. It lists the default settings and describes optional settings. Even though the jumpers and DIP switches are preset during board assembly, verify that each one is in the position that works best with your application. Figure 6-1 illustrates the physical locations of the jumper and DIP switches.

If you are going to use the 8x930Ax firmware, refer to“Using the on-board 8x930Ax firmware” on page 2-9 for configuration details, including driver information and jumper and DIP switch settings.

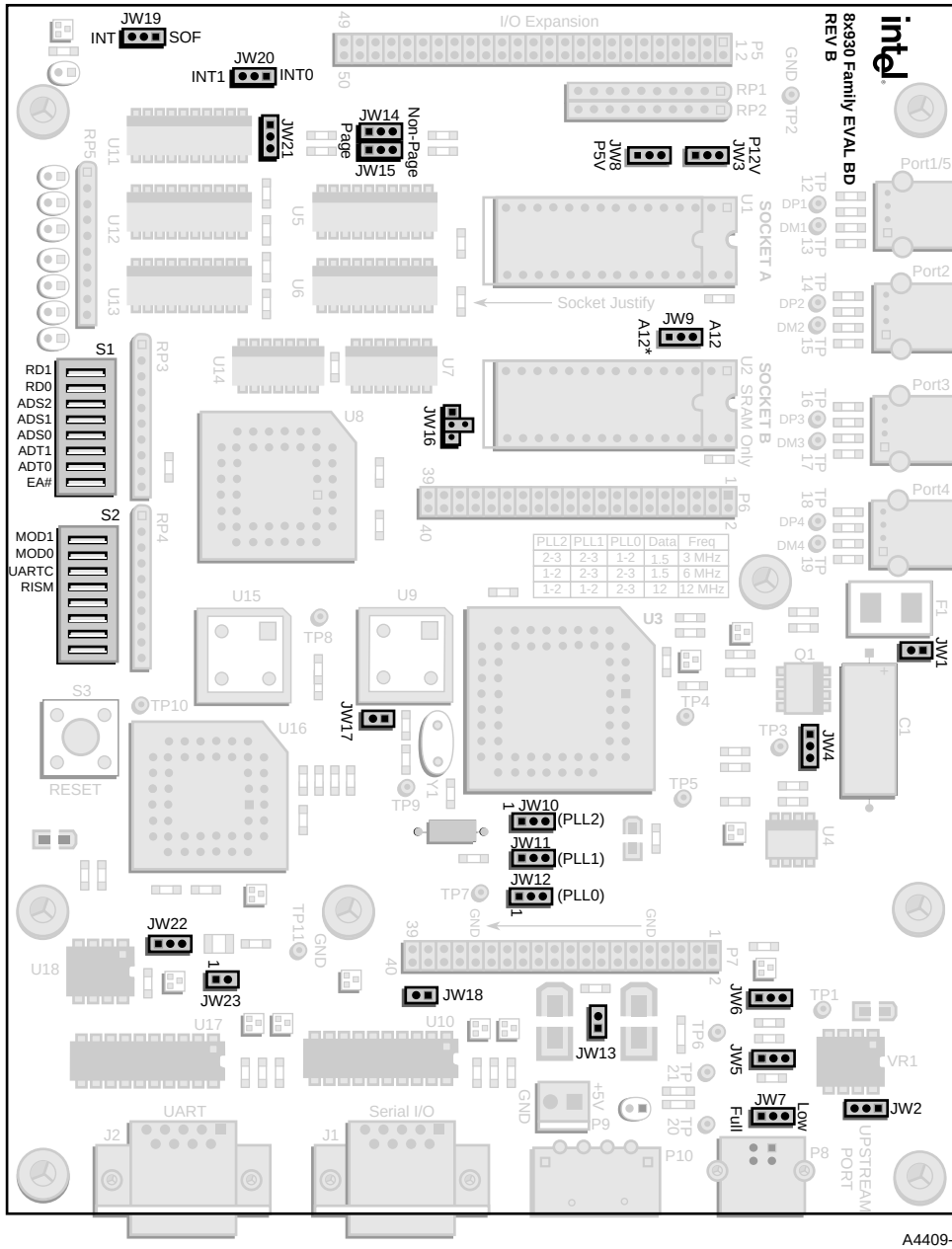


Figure 6-1. 8x930 Family USB Evaluation Board's Jumper and DIP-Switch Locations

6.1 EVALUATION BOARD JUMPER SETTINGS

This section discusses the evaluation board's jumper settings.

6.1.1 Default Jumper Settings

The evaluation board's default jumper settings are listed in Table 6-1.

Table 6-1. Evaluation Board's Default Jumper Settings

Jumper	Setting	Jumper	Setting
JW1	open	JW13	short
JW2	2–3	JW14	1–2
JW3	2–3	JW15	1–2
JW4	1–2	JW16	2–4
JW6	1–2	JW17	short
JW7	1–2	JW18	short
JW8	2–3	JW19	2–3
JW9	2–3	JW20	1–2
JW10	1–2	JW21	1–2
JW11	1–2	JW22	2–3
JW12	2–3	JW23	short

6.1.2 JW1: Bypassing the Downstream-Port Power Fuse

Use JW1 if you want to bypass the downstream-port power fuse. Bypassing this power fuse can reduce the voltage drop when the board is configured to be a bus-powered hub.

6.1.3 JW2: Connecting Power

Use JW2 to set power connections. See Table 6-2 below for specific settings.

Table 6-2. JW2: Power Connections

Setting	Description
1–2	Connects the input power from the upstream USB port to P5V. Also needed for bus-powered function or hub connection.
2–3	Allows the upstream power to be disconnected from P5V.

6.1.4 JW3: Programming Voltage for Flash Memory Devices

Use JW3 to connect P12V to socket A to program voltage for flash memory devices. When 1–2 is jumpered, 5 volt is applied. When 2–3 is jumpered, JW3 connects P12V to socket A, pin 1.

6.1.5 JW4: Powering Downstream Ports

Use JW4 to power downstream ports. See Table 6-3 below for specific settings.

Table 6-3. JW4: Downstream Port Powering

Setting	Description
1–2	Allows you to use UPWEN# to switch the power on and off to the downstream ports. This normally occurs on the bus-powered hubs.
2–3	Connects P5v directly to the downstream ports.

6.1.6 JW5: Controlling the Pullup Resistor Source on the Upstream Port

Use JW5 to control the source for the pullup resistor on the upstream port. See Table 6-4 for specific settings.

Table 6-4. JW5: Controlling the Pullup Resistor Source on the Upstream Port

Setting	Description
1–2	Pullup is created by a Thevenin equivalent circuit for a 1.5k Ω resistor to 3.3V regulated source.
2–3	The pullup is a 1.5k Ω resistor to a 3.3V regulated source.

6.1.7 JW6–JW9 Jumper Settings

Set JW6–JW9 according to the information given in Table 6-5.

Table 6-5. JW6–JW9 Jumper Settings

Jumper	Description
JW6 (SHDN#)	Enables RESET to control the pullup on the USB. **1–2 shunted**: When 1–2 is shunted, pullup is **always** enabled. During a reset, the device stays electrically connected. **2–3 shunted**: When 2–3 is shunted, RESET disables the pullup for either LOW or FULL speed rates on the USB. During a reset, the device will simulate an electrical disconnect followed by an electrical connect. This means that the client driver will be removed from memory and then reloaded upon reconnect.
JW7 (LOW/FULL)	Connects a 1.5 KΩ resistor to either D_P or D_M depending upon the desired rate selection upon connection to the USB. LOW position jumpered selects 1.5 Mbps. FULL position jumpered selects 12 Mbps.
JW8 (P5V/A17)	Selects either V_{CC} or A17 for socket A, pin 30. A17 should be connected to pin 20 only when a 256-Kbyte flash EPROM is used.
JW9	Allows you to invert address line A12 to Socket A (ROM). **A12 position**: If you shunt the A12 position, it connects A12 directly to the socket. **A12* position**: If you shunt A12 before the socket, it allows you to swap each 8K-byte block of the EPROM. This allows you to place code in these alternate blocks. When you move this jumper, you can map the reset vector address for boot. **Only Socket A is affected**. Refer to Chapter 2, “Getting Started” for more details.

6.1.8 JW10–JW12: Setting Core Operating Frequency and USB Data Rate

Table 6-6 shows the three, three-pin jumpers that select the core operating frequency and the USB data rate.

Table 6-6. JW10–JW12: Core Operating Frequency and USB Data Rate Settings

PLLSEL2 JW10	PLLSEL1 JW11	PLLSEL0 JW12	USB Data Rate	Core Frequency	XTAL
2–3	2–3	1–2	1.5 Mbps	3 MHz	6 MHz
1–2	2–3	2–3	1.5 Mbps	6 MHz	12 MHz
1–2	1–2	2–3	12 Mbps	12 MHz	12 MHz

6.1.9 JW13: Installed Jumpers

JW13 is one of five installed jumpers. These jumpers are jumpered in track on the board. For more information about modifying this jumper, see the schematic diagram in Appendix C, "Schematics," or contact your Intel field-application engineer.

6.1.10 JW14 and JW15: Setting the Page/Nonpage Mode

JW14 and JW15 are also known as page and nonpage. Use these jumpers to select the page/nonpage mode for routing the data bus from the upper-address port (during page mode) or from the lower-address port (during nonpage mode). Table 6-7 shows the settings.

Table 6-7. Setting Page/Nonpage Mode

JW14 and JW15	Mode
Both 1–2	nonpage
Both 2–3	page
Any other setting not allowed	

6.1.11 JW16–JW23: Setting the Remaining Jumpers

Set the remaining jumpers according to the information in Table 6-8.

Table 6-8. JW16–JW23: Setting the Remaining Jumpers

Jumper	Description
JW16	Allows the Thevenin pullup for the upstream USB port to be controlled via software. When 1-2 is shunted. P3.3 controls the pullup. Setting P3.3 enables the pullup. Clearing P3.3 disables the pullup.
JW17	JW17 is one of five installed jumpers that are jumpered in track on the board. JW17 allows you to disconnect the clock oscillator from the evaluation board so that you can use a crystal. To use the crystal, you must either cut JW17 or remove the clock oscillator. **Note:** If you select a USB data rate of 1.5 MB per second (Mbps), with a 3 MHz core, you **must** install a 6 MHz crystal on the board.
JW18	JW18 is one of five installed jumpers. These jumpers are jumpered in track on the board. For more information about modifying this jumper, see the schematic diagram Appendix C, "Schematics," or contact your Intel field-application engineer.
JW19 (SOF#/INT1#)	Selects source of INT1# for the 8x930 P3.3/INT1# pin. You can enable the UART to use INT1# when INT1# is shunted.
JW20 (INT0#/INT1#)	Selects the interrupt input the UART will use. When you select INT1, set JW20 to INT1#.

Table 6-8. JW16–JW23: Setting the Remaining Jumpers (Continued)

Jumper	Description
JW21	JW21 is one of five installed jumpers. These jumpers are jumpered in track on the board. For more information about modifying this jumper, see the schematic diagram Appendix C, “Schematics,” or contact your Intel field-application engineer.
JW22	JW22 is one of five installed jumpers. These jumpers are jumpered in track on the board. For more information about modifying this jumper, see the schematic diagram Appendix C, “Schematics,” or contact your Intel field-application engineer.
JW23	JW23 is one of five installed jumpers. These jumpers are jumpered in track on the board. For more information about modifying this jumper, see the schematic diagram Appendix C, “Schematics,” or contact your Intel field-application engineer.

6.2 EVALUATION BOARD DIP-SWITCH SETTINGS

This section discusses the evaluation board’s DIP-switch settings and lists them in Table 6-9.

Table 6-9. Evaluation Board’s Default DIP-Switch Settings

Switch	Position
RD1	on
RD0	on
ADS2	on
ADS1	on
ADS0	off
ADT1	on

Switch	Position
ADT0	on
EA#	on
MOD1	on
MOD0	off
UARTC	on
RISM	on

6.2.1 RD1 and RDO: Setting External Memory Size

Use switches RD1 and RD0 to select the external memory size (see Table 6-10). These settings **must** match the RD1 and RD0 bits in UCONFIG0 (refer to the *8x930Ax, 8x930Hx Universal Serial Bus Microcontroller User's Manual* for UCONFIG0 settings).

Table 6-10. External Memory Size Settings

RD1	RD0	External Address Size
off	off	64K 8051 compatible
off	on	64K byte
on	off	128K byte
on	on	256K byte

6.2.2 ADS2, ADS1, and ADS0: Setting Socket A's Device Size

Use switches ADS2, ADS1, and ADS0 to select Socket A's device size (see Table 6-11). Figure 6-1 shows the location of Socket A.

Table 6-11. Socket A's Device Size Settings

ADS2	ADS1	ADS0	Socket A Device Size
on	on	on	8K byte
on	on	off	32K byte
on	off	on	64K byte
on	off	off	128K byte
off	on	on	256K byte
off	on	off	reserved
off	off	on	reserved
off	off	off	reserved

6.2.3 ADT1 and ADT0: Setting Socket A's Device Type

Use switches ADT1 and ADT0 to select socket A's device type (see Table 6-12).

Table 6-12. Socket A's Device Type Settings

ADT1	ADT0	Socket A Device Type
on	on	EPROM
on	off	SRAM
off	on	Bulk Flash EPROM
off	off	Boot Block Flash EPROM

6.2.4 EA#: Setting Internal or External ROM Addresses

Use switch EA# to select either internal or external ROM addressing (see Table 6-13). The evaluation board does not have internal memory, so this switch is normally left in the "on" position.

Table 6-13. Internal or External ROM Address Settings

EA#	ROM Location
off	internal ROM
on	external ROM

6.2.5 M0D1 and M0D0: Setting Modes

Use the mode switches (MOD1 and MOD0) to select the mode for the evaluation board with respect to page versus nonpage and binary versus source (see Table 6-14).

If the board uses the 32-Kbyte EPROM installed from the factory, it contains four versions of RISM in each 8-Kbyte block of the device. The mode switches select a version of RISM that runs the board in the selected mode. When selecting between page and nonpage modes, set the jumpers, JW14 and JW15, to match the mode. JW14 and JW15 are located near socket A.

Table 6-14. MODE Settings

MOD1	MOD0	Memory	Addressing MODE
on	on	source	nonpage
on	off	binary	nonpage
off	on	source	page
off	off	binary	page

6.2.6 UARTC: Enabling the UART

Use the UARTC switch to enable the UART (see Table 6-15). The switch disconnects the chip select from the UART, so RISM cannot detect the presence of the UART.

Table 6-15. UART Enabling Settings

UARTC	UART state
on	enabled
off	disabled

6.2.7 RISM: Allocating Memory Space

Use the RISM switch to select the configuration of memory space allocation (see Table 6-16).

Table 6-16. Memory Space Allocation Settings

RISM	Socket Configuration
off	Socket A maps to FE and FF pages; and RAM maps to 00 and 01 pages. UART is disabled.
on	Socket A maps bottom 8K of device to FF:0000 and FF:E000 blocks; and RAM maps to FF:2000 to FF:DFFF.

NOTE

If the RISM switch is “off”, the UART is disabled, and the upper-memory area is a contiguous block of ROM.

A

Components and Connectors

APPENDIX A
COMPONENTS AND CONNECTORS

This appendix contains the following items:

- Figure A-1 on page A-2 which labels the major board components and connectors.
- Table A-1 on page A-3 which lists all 8x930 Family USB evaluation board components.
- Table A-2 on page A-5 which describes the serial port connector and necessary cable for connecting the evaluation board to the host PC.
- Figure A-2 on page A-6 which shows how to assemble a 25-pin to 9-pin serial port interface adapter cable to attach to the evaluation board.

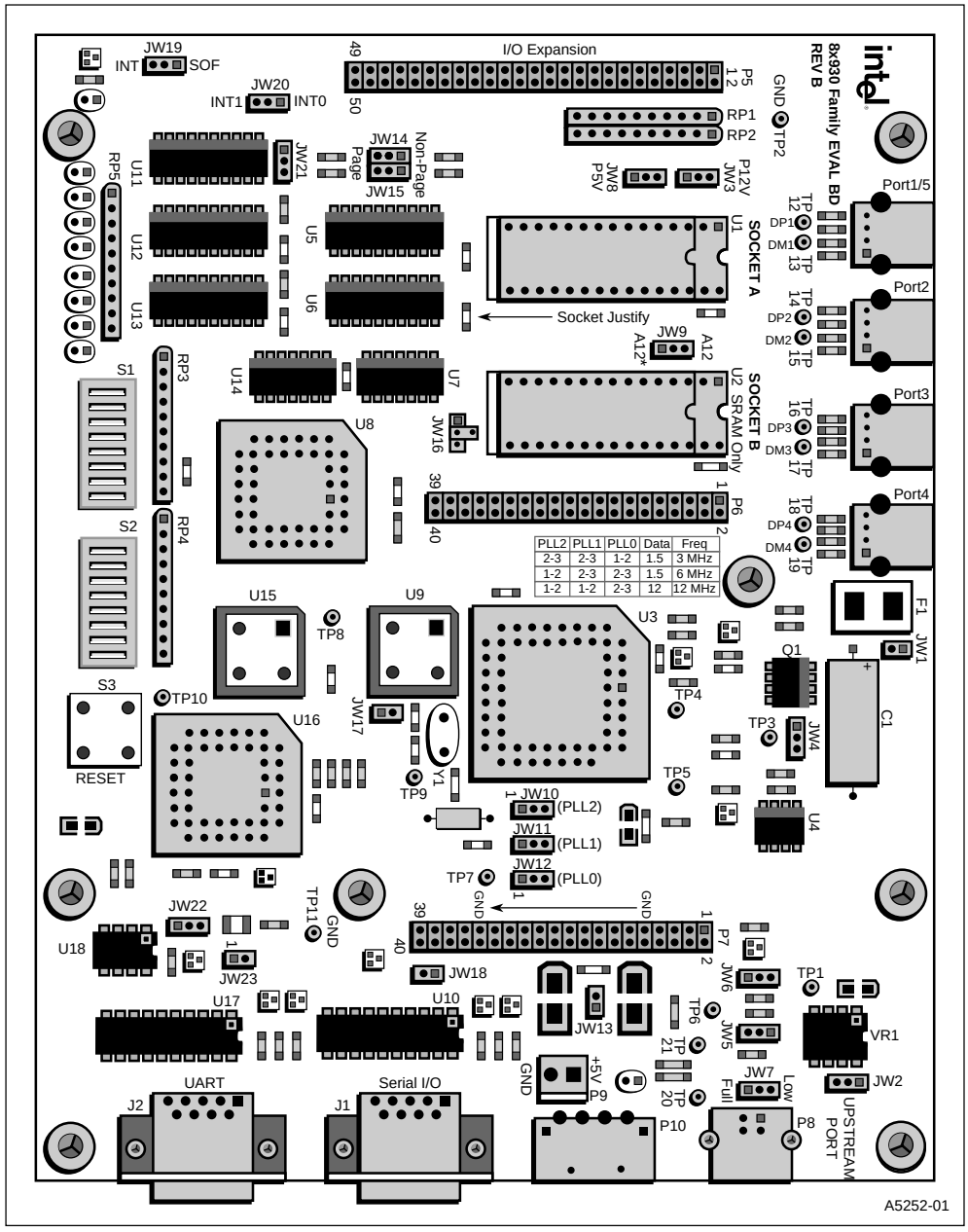


Figure A-1. 8x930 Family USB Evaluation Board Diagram

A.1 COMPONENT LIST

Table A-1 lists all 8x930 Family USB evaluation board components.

Table A-1. Components List

Item	Quantity	Designator	Part	Manufacturer
1	19	C3-C7, C9, C10, C12, C13, C15-C18, C22-C26, C28	0.1 uF	Kemet C1206C104M5UAC
2	1	C27	0.33 uF	Kernet C1210C334M5UAC
3	2	C2, C11	1 uF	Kemet T491A105K016AS
4	2	C8, C14	10 uF	Kemet T491C106K016AS
5	3	C19-C21	30 pF	Kemet C0805C330J5GAC
6	1	C1	150 uF solid tant	Mallory TXA157M010P1F
7	1	U15	1.8432 MHz	Fox H5C-2 tri-state output
8	1	U9	12.000 MHz	Fox H5C-2 tri-state output
9	1	P9	CONN 2POS	Berg
10	1	P10	CONN 4POS	AMP 641737-1
11	2	J1, J2	CONN 9SUBD	Amphenol 617-C009S-BF220
12	2	P6, P7	CONN 40POS	Berg 2x20 0.230" pin length
13	1	P5	CONN 50POS	Berg 2x25 0.230" pin length
14	4	P1, P2, P3, P4	CONN USB type A	AMP 787616-1
15	1	P8	CONN USB type B	AMP 95-4042-1 proposal #
16	2	CR2,CR4	FMMD914	ZETEX FMMD914CT-ND
17	2	CR1, CR3	DIODE LED	MicroLamps MLED-934GD
18	8	CR5-CR12	DIODE LED	MicroLamps MLED-934RD
19	1	F1	SMD200	Raychem SMD200
20	1	U16	16C550	T.I. TL16C550AFN
21	1	U11	74HC240	Motorola MC74HC240ADW
22	2	U5, U6	74HCT245	Motorola MC74HCT245ADW
23	2	U12, U13	74AC373	Motorola MC74AC373AD
24	1	U7	74HC132	Motorola MC74HC132D
25	1	U14	74HC30	Motorola MC74HC30D
26	1	U3	82930A	Intel 8x930Ax
27	1	U8	EPM7032	Altera EPM7032LC44-15
28	2	U10, U17	MAX233	Maxim MAX233CPP
29	1	U18	MAX708	Maxim MAX708CPA
30	1	U2	128k SRAM	Hitachi HM628128LP-10

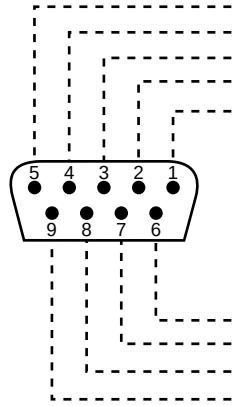
Table A-1. Components List

Item	Quantity	Designator	Part	Manufacturer
31	1	U1	32k EPROM	27C256-12 Ceramic windowed
32	1	U4	TLC27L2	T.I. TLC27L2CD
33	1	L1	47 uH	Dale IM-2 47uH
34	13	JW2, JW3, JW6-JW8, JW9, JW10-JW12, JW14, JW15, JW19, JW20	JUMPER 3PIN	Berg 0.230" pin length
35	1	VR1	LT1121-3.3	Linear Tech LT1121CN8-3.3
36	3	R19, R48, R57	470k ohm	Dale CRCW1206
37	8	R1, R4, R5, R8, R9, R12, R13, R16	15k ohm	Dale CRCW1206
38	5	R24, R30, R36, R51, R56	1.5k ohm	Dale CRCW1206
39	1	R28	2.2k ohm	Dale CRCW1206
40	5	R17, R22, R25, R26, R29	4.7k ohm	Dale CRCW1206
41	22	R18, R20, R21, R23, R27, R34, R35, R37-R47, R49, R50, R54	10k ohm	Dale CRCW1206
42	3	R33, R52, R53	270 ohm	Dale CRCW1206
43	10	R2, R3, R6, R7, R10, R11, R14, R15, R31, R32	33 ohm	Dale CRCW1206
44	1	RP5	270 ohm	Dale 10A1 271G
45	2	RP3, RP4	10k ohm	Dale 10A1 103G
46	2	RP1, RP2	33k ohm	Dale 10A1 333G
47	2	U9, U12	32-PIN DIP socket	Augat 832-AG11D-ES
48	2	U8, U16	44-pin PLCC socket	Augat PCS-044A-1
49	1	U3	68-pin PLCC socket	Augat PCS-068A-1
50	1	S3	SWITCH MOM	ITT Canon KSAOM211
51	2	S1, S2	SWITCH DIP8	Alcoswitch GDS08
52	1	Q1	Si9433DY	Temic Si9433DY
53	5	Q2, Q5, Q6, Q8, Q10	MMBT2907ALT1	Motorola MMBT2907ALT1
54	4	Q3, Q4, Q7, Q9	MMBF170LT1	Motorola MMBF170LT1
55	11	TP1-TP11	TEST POINT	Mil-Max 3132-0-00-15-00-00-08-0
56	1	S3	key cap	ITTCanon K0101
57	13	Jumper Shunts	shunt 2 pin	

A.2 HOST SERIAL CONNECTOR DESCRIPTION

Table A-2 shows a 9-pin serial connector and the connection of the signals to the evaluation board. This connector is a DB-9S, RS-232 (DB9); it is included as part of the evaluation board's development kit. The cable must connect all nine signal lines, without loopbacks or crossed wires. The evaluation board port P1 is pinned out for a standard IBM-PC/AT* type serial port.

Table A-2. P1 Host Serial Connector

P1 Connector	Pin Nos.	Host RS-232 Signal Name/Function	Connection on Evaluation Board
	5	SG – Signal Ground	V_{SS}
	4	DTR – Data Terminal Ready	INIT
	3	TXD – Transmit Data	RXD
	2	RXD – Receive Data	TXD
	1	DCD – Data Carrier Detect	INIT
	6	DSR – Data Set Ready	INIT
	7	RTS – Request To Send	CTS (pin 8)
	8	CTS – Clear To Send	RTS (pin 7)
	9	RI – Ring Indicator	Run Indicator

If your computer has a 25-pin serial port connector, we recommend you buy a standard RS-232 25-pin to 9-pin conversion adapter or cable. Figure A-2 on page A-6 shows you how to assemble a 25-pin to 9-pin serial port interface adapter cable for the correct connection to the evaluation board.

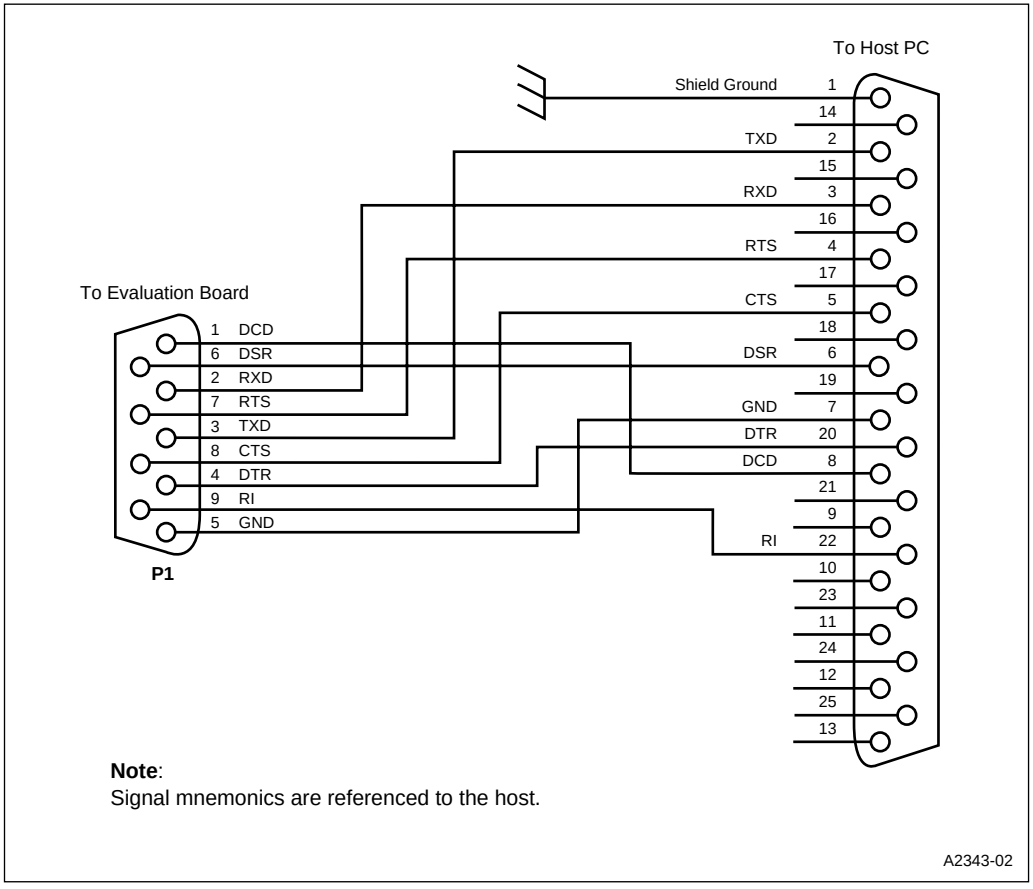


Figure A-2. Serial Interface

B

CPLD Equations

APPENDIX B
CPLD EQUATIONS

This appendix lists the CPLD equations.

```
MODULE u8pal
TITLE ’8x930xx USBM Eval. Bd. Memory Configuration CPLD’

" Last Modified: 10/26/96

" declare inputs

 a17,a16,a15,a14,a13,wr  PIN  9,12,1,4,2,11;
 psen,ale,memdisabl,alonePIN  13,14,7,8;
 rd1,rd0,ads2,ads1,ads0  PIN  16,17,18,19,20;
 adt1,adt0,ea,mode1,mode0PIN  21,24,25,26,27;
 uartaddr,clk12mhz,oe    PIN  31,43,44;

" declare outputs

 rama16,ramcs,ramoe,romcsPIN  5,41,40,39;
 roma18,roma16,roma15    PIN  32,33,34;
 roma14,roma13,romoe     PIN  36,37,38;
 ale_active,uartnabl     PIN  29,28;
 membufen                PIN  6;

 q4..q0                  NODE ISTYPE    ’REG’;
 rst_ale_cnt             NODE;

" declare contants

 x,c,l,h,z     =    .X.,.C.,0,1,.Z.;
```

```
 on,off        =    0,1;
"The memory size is determined by the 2 switches, rd1 and rd0 which correspond to
"the 2 bits in CONFIG0.  They determine if there are 18 bits, 17 bits or 16 bits
"of external addressing.

 mem_size      =                  [rd1, rd0];
 mem_size_256k =    (mem_size ==   [on,  on ]);
 mem_size_128k =    (mem_size ==   [on,  off]);
 mem_size_64k  =    (mem_size ==   [off, on ]);
 mem_size_8051 =    (mem_size ==   [off, off]);

"The memory type for the ROM socket is selected with adt1 and adt0.

 rom_mem_type  =                       [adt1, adt0];
 eprom         =    (rom_mem_type==    [on,   on  ]);
 sram          =    (rom_mem_type==    [on,   off ]);
 flash         =    (rom_mem_type==    [off,  on  ]);
 boot_block    =    (rom_mem_type==    [off,  off ]);

"The physical size for the ROM socket device is selected with ads2,ads1,and ads0.

 rom_size      =                       [ads2, ads1, ads0];
 rom_256k      =    (rom_size ==       [off,  on,   on  ]);
 rom_128k      =    (rom_size ==       [on,   off,  off ]);
 rom_64k       =    (rom_size ==       [on,   off,  on  ]);
 rom_32k       =    (rom_size ==       [on,   on,   off ]);
 rom_8k        =    (rom_size ==       [off,  on,   off ]);

"Mode is merely used to map a particular block to reside at the reset vector
"address and a block to reside at the reset vector address.  The mode of
"addressing is selected with mode1 and mode0 and they correspond to DIP
"switches. The 4 available modes for the 251 are binary paged, binary non-paged,
"source paged, and source non-paged.  The addressing of the ROM socket is
"controlled with the mode selection in which a specific 8k byte block of the ROM
```

```
"socket is selected.  The block will contain a RISM and CONFIG0 and CONFIG1
"value that will initialize the 251 with the selected mode.
mode      =                  [ mode1, mode0 ];
mode_bn        =    (mode     ==   [    on,   off ]);
mode_bp        =    (mode     ==   [   off,   off ]);
mode_sn        =    (mode     ==   [    on,    on ]);
mode_sp        =    (mode     ==   [   off,    on ]);


"      32k byte EPROM physical mapping


"  ---------------------------------------
" |   251 binary non-paged config bytes   |
" |                                       |
" |   251 binary paged code               |
"  ---------------------------------------   6000H
" |   251 source non-paged config bytes   |
" |                                       |
" |   251 source paged code               |
"  ---------------------------------------   4000H
" |   251 source paged config bytes       |
" |                                       |
" |   251 source non-paged code           |
"  ---------------------------------------   2000H
" |   251 binary paged config bytes       |
" |                                       |
" |   251 binary non-paged code           |
"  ---------------------------------------   0000H


"The following defines blocks of addresses for each memory size.  The blocks
"partition the memory is a way such that the RAM socket can be mapped into the
"upper memory space (ROM space).  RAM in the upper space allows debugging of
"code to take place in the normally defined code region.

 address_256k  =    [a17,a16, a15,a14,a13,x,x,x,x,x,x,x,x,x,x,x,x];
 addrblk_1_256k=    ((address_256k >= ^h000000) & (address_256k <= ^h01ffff));
 addrblk_2_256k=    ((address_256k >= ^hfe0000) & (address_256k <= ^hfeffff));
 addrblk_3_256k=    (address_256k == ^hff0000);
 addrblk_4_256k=    ((address_256k >= ^hff2000) & (address_256k <= ^hffdfff));
 addrblk_5_256k=    (address_256k == ^hffe000);
```

```
 address_128k  =    [a16,a15,a14,a13,x,x,x,x,x,x,x,x,x,x,x,x];
 addrblk_1_128k=    ((address_128k >= ^h000000) & (address_128k <= ^h00ffff));
 addrblk_2_128k=    (address_128k == ^hff0000);
 addrblk_3_128k=    ((address_128k >= ^hff2000) & (address_128k <= ^hffdfff));
 addrblk_4_128k=    (address_128k == ^hffe000);

 address_64k   =    [a15,a14,a13,x,x,x,x,x,x,x,x,x,x,x,x,x];
 addrblk_1_64k =    (address_64k == ^h0000);
 addrblk_2_64k =    ((address_64k >= ^h2000) & (address_64k <= ^hdfff));
 addrblk_3_64k =    (address_64k == ^he000);

"The external UART (16C550) is mapped into the memory in the uppermost region
"below the configuration bytes.  The addressing is fixed as follows:
"64k address spaceffc0 - ffdf (RAM in 8051 compatible mode)
"128k address spacefffc0 - fffdf
"128k address spacefffc0 - fffdf

 uart_64k      =    (address_64k  == ^hffff) & !uartaddr;
 uart_128k     =    (address_128k == ^hffffff) & !uartaddr;
 uart_256k     =    (address_256k == ^hffffff) & !uartaddr;

 ale_cnt       =    [q3..q0];
 old_ale       =    q4;
 ale_not_act   =    (ale_cnt == [1,1,1,1]);
 memdisabled   =    (memdisabl == l);
EQUATIONS

 !ramcs = addrblk_1_256k & mem_size_256k & !memdisabled;
 !ramcs = addrblk_2_256k & mem_size_256k & !memdisabled & !alone;
 !ramcs = addrblk_4_256k & mem_size_256k & !memdisabled & !alone;

 !ramcs = addrblk_1_128k & mem_size_128k & !memdisabled;
 !ramcs = addrblk_3_128k & mem_size_128k & !memdisabled & !alone;
```

```
 !ramcs = addrblk_2_64k & mem_size_64k & !memdisabled & !alone;

 !ramcs = addrblk_2_64k & mem_size_8051 & !memdisabled & !uart_64k & !alone;


 !romcs = addrblk_3_256k & mem_size_256k & !memdisabled;
 !romcs = addrblk_5_256k & mem_size_256k & uartaddr & !memdisabled;

 !romcs = addrblk_2_128k & mem_size_128k & !memdisabled;
 !romcs = addrblk_4_128k & mem_size_128k & uartaddr & !memdisabled;

 !romcs = addrblk_1_64k & mem_size_64k & !memdisabled;
 !romcs = addrblk_3_64k & mem_size_64k & uartaddr & !memdisabled;

 !romcs = addrblk_1_64k & mem_size_8051 & !memdisabled;
 !romcs = addrblk_3_64k & mem_size_8051 & uartaddr & !memdisabled;

 !romcs = addrblk_2_256k & mem_size_256k & alone & !memdisabled;
 !romcs = addrblk_4_256k & mem_size_256k & alone & !memdisabled;
 !romcs = addrblk_5_256k & mem_size_256k & alone & !memdisabled;

 !romcs = addrblk_3_128k & mem_size_128k & alone & !memdisabled;
 !romcs = addrblk_4_128k & mem_size_128k & alone & !memdisabled;

 !romcs = addrblk_2_64k & mem_size_64k & alone & !memdisabled;
 !romcs = addrblk_3_64k & mem_size_64k & alone & !memdisabled;

 !romcs = addrblk_2_64k & mem_size_8051 & alone & !memdisabled;

" ramoe
      WHEN (mem_size_256k # mem_size_128k # mem_size_64k) THEN !ramoe = !psen
 ELSE WHEN mem_size_8051 THEN !ramoe = (!a16 # !psen);
```

```
" rama16
      WHEN (mem_size_256k # mem_size_128k) THEN rama16 = a16
 ELSE rama16 = l;

"Here roma13 and roma14 are passed through or inverted based upon the mode
"switch settings.  A particular 8k block is mapped to the reset vector and a 8k
"block to the config byte block.

" roma13
      WHEN rom_8k THEN roma13 = h
 ELSE {
     roma13 = a13 & !alone & (mode_bn # mode_sp);
     roma13 = !a13 & !alone & (mode_sn # mode_bp);
     }
      WHEN alone THEN roma13 = a13;

"roma14
      WHEN sram THEN roma14 = wr
 ELSE {
     roma14 = a14 & !alone & (mode_bn # mode_sn);
     roma14 = !a14 & !alone & (mode_sp # mode_bp);
     }
      WHEN alone THEN roma14 = a14;

" roma15
      WHEN sram THEN roma15 = a14
 ELSE roma15 = a15;

" roma16
      WHEN (mem_size_256k # mem_size_128k) THEN roma16 = a16
 ELSE roma16 = l;
" roma18
      WHEN (rom_256k # rom_128k) & !(flash # boot_block)
        THEN roma18 = !a15
```

```
 ELSE WHEN (flash # boot_block) THEN !roma18 = !wr
 ELSE roma18 = l;

 !uartnabl=    uart_64k & (mem_size_64k # mem_size_8051) & !alone;
 !uartnabl=    uart_128k & mem_size_128k & !alone;
 !uartnabl=    uart_256k & mem_size_256k & !alone;

" !membufen
      WHEN (mem_size_256k # mem_size_128k # mem_size_64k) & !memdisabled
        THEN !membufen = (!psen # !wr)
 ELSE WHEN mem_size_8051 & !memdisabled THEN !membufen = (!a16 # !psen # !wr);

"ale_cnt is a 4 bit up counter that counts clocks between ale edges.  If an ale
"edge doesn't occur within 16 clocks of the previous edge, ale_active will be
"untrue and the external output will turn off.  As long as an edge occurs within
"the 16 clocks ale_active will remain true and the LED will stay on.

 ale_cnt.clk=clk12mhz;
 old_ale.clk=clk12mhz;
 old_ale  :=   ale;
 rst_ale_cnt=ale $ old_ale; "($ = XOR)
 ale_cnt.ar=rst_ale_cnt;

      WHEN ale_not_act THEN ale_cnt := ale_cnt
 ELSE ale_cnt := ale_cnt + 1;
 ale_active=   !ale_not_act;
rama16.oe      =    !oe;
 ramcs.oe      =    !oe;
 ramoe.oe      =    !oe;
 roma13.oe     =    !oe;
 roma14.oe     =    !oe;
 roma15.oe     =    !oe;
 roma16.oe     =    !oe;
 roma18.oe     =    !oe;
```

```
 romcs.oe      =    !oe;
 ale_active.oe =    !oe;
 uartnabl.oe   =    !oe;
 membufen.oe   =    !oe:

TEST_VECTORS 'Memory addressing'

([address_256k,mem_size,rom_size,memdisabl,uartaddr] -> [romcs,ramcs,uartnabl])
"vector
 [    ^hff0000,       0,       0,        1,       1] -> [    l,    h,       h];"1
 [    ^hffffff,       0,       0,        1,       1] -> [    l,    h,       h];"2
 [    ^hff0000,       1,       2,        1,       1] -> [    l,    h,       h];"3
 [    ^hff0000,       0,       0,        1,       1] -> [    l,    h,       h];"4
 [    ^h000000,       3,       0,        1,       1] -> [    l,    h,       h];"5
 [    ^hffffff,       0,       0,        1,       0] -> [    h,    h,       l];"6
 [    ^hffffff,       1,       0,        1,       0] -> [    h,    h,       l];"7
 [    ^hffffff,       2,       0,        1,       0] -> [    h,    h,       l];"8
 [    ^hffffff,       3,       0,        1,       0] -> [    h,    h,       l];"9
 [    ^hffffff,       3,       0,        0,       1] -> [    h,    h,       h];"10
 [    ^hff0000,       3,       0,        0,       1] -> [    h,    h,       h];"11
 [    ^h000000,       3,       0,        0,       1] -> [    h,    h,       h];"12
 [    ^h010000,       3,       0,        1,       1] -> [    l,    h,       h];"13
 [    ^h0f0000,       3,       0,        1,       1] -> [    l,    h,       h];"14
 [    ^h000000,       0,       0,        1,       1] -> [    h,    l,       h];"15
 [    ^h01ffff,       0,       0,        1,       1] -> [    h,    l,       h];"16
 [    ^hffe000,       0,       0,        1,       1] -> [    l,    h,       h];"17
 [    ^hffdfff,       0,       0,        1,       1] -> [    h,    l,       h];"18
 [    ^hff8000,       0,       0,        1,       1] -> [    h,    l,       h];"19
 [    ^hffd000,       0,       3,        1,       1] -> [    h,    l,       h];"20
 [    ^hff8fff,       0,       2,        1,       1] -> [    h,    l,       h];"21
 [    ^h00ffff,       1,       0,        1,       1] -> [    h,    l,       h];"22
TEST_VECTORS 'output enables'

([mem_size,psen,a16] -> [ramoe])
```

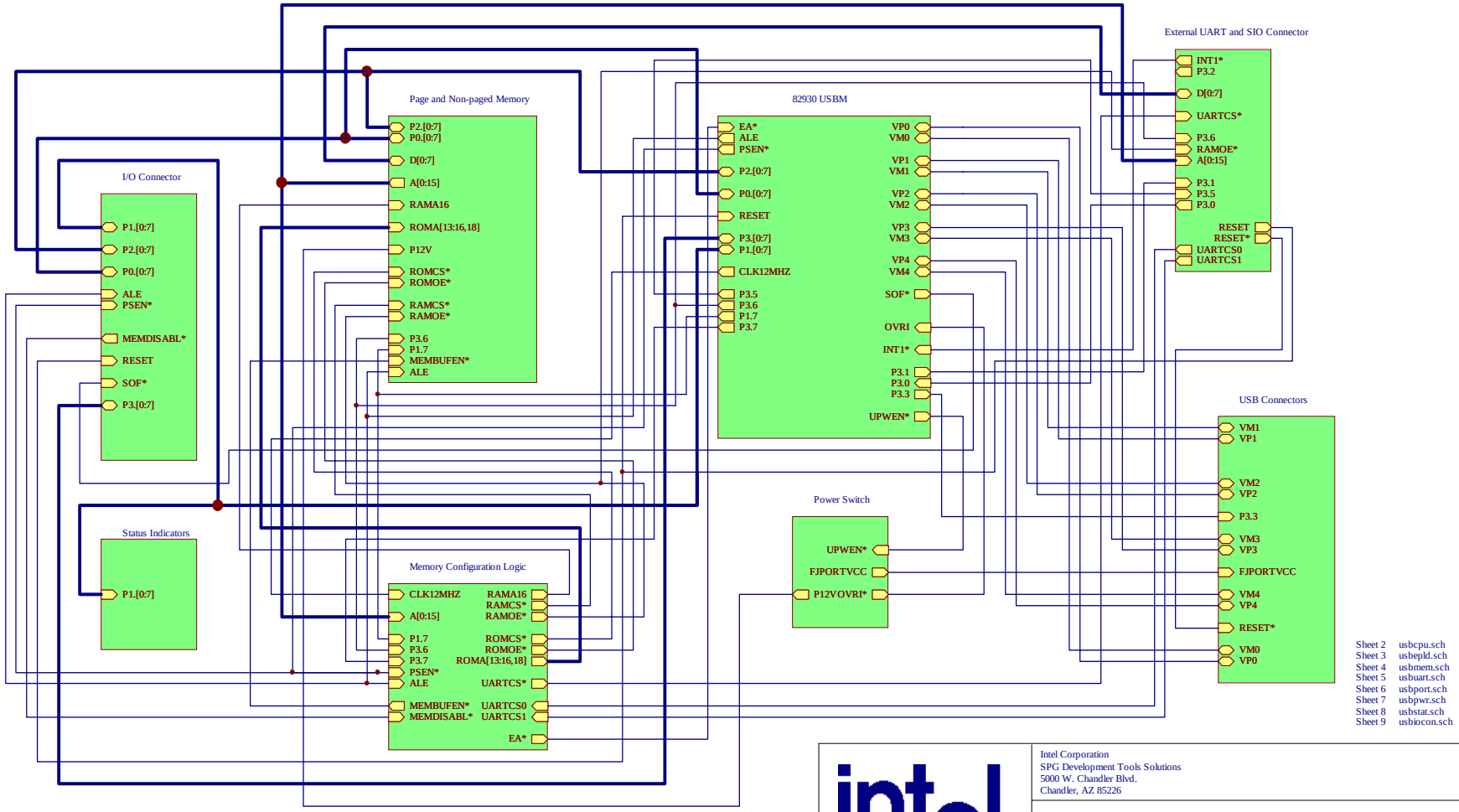
```
[       0,   1,  1] -> [    h];"23
[       0,   0,  1] -> [    l];"24
[       0,   1,  0] -> [    h];"25
[       3,   1,  0] -> [    l];"26
[       1,   1,  1] -> [    h];"27
[       2,   0,  1] -> [    l];"28
[       2,   0,  0] -> [    l];"29
[       2,   1,  0] -> [    h];"30
[       3,   1,  0] -> [    l];"31
[       3,   0,  1] -> [    l];"32

END
```

C

Schematics

APPENDIX C
SCHEMATICS

Use the schematics in this chapter to set up or change the jumpers on the board and to interface the board to your application prototype. This schematic includes all the post-fabrication modifications that may be apparent on the evaluation board.



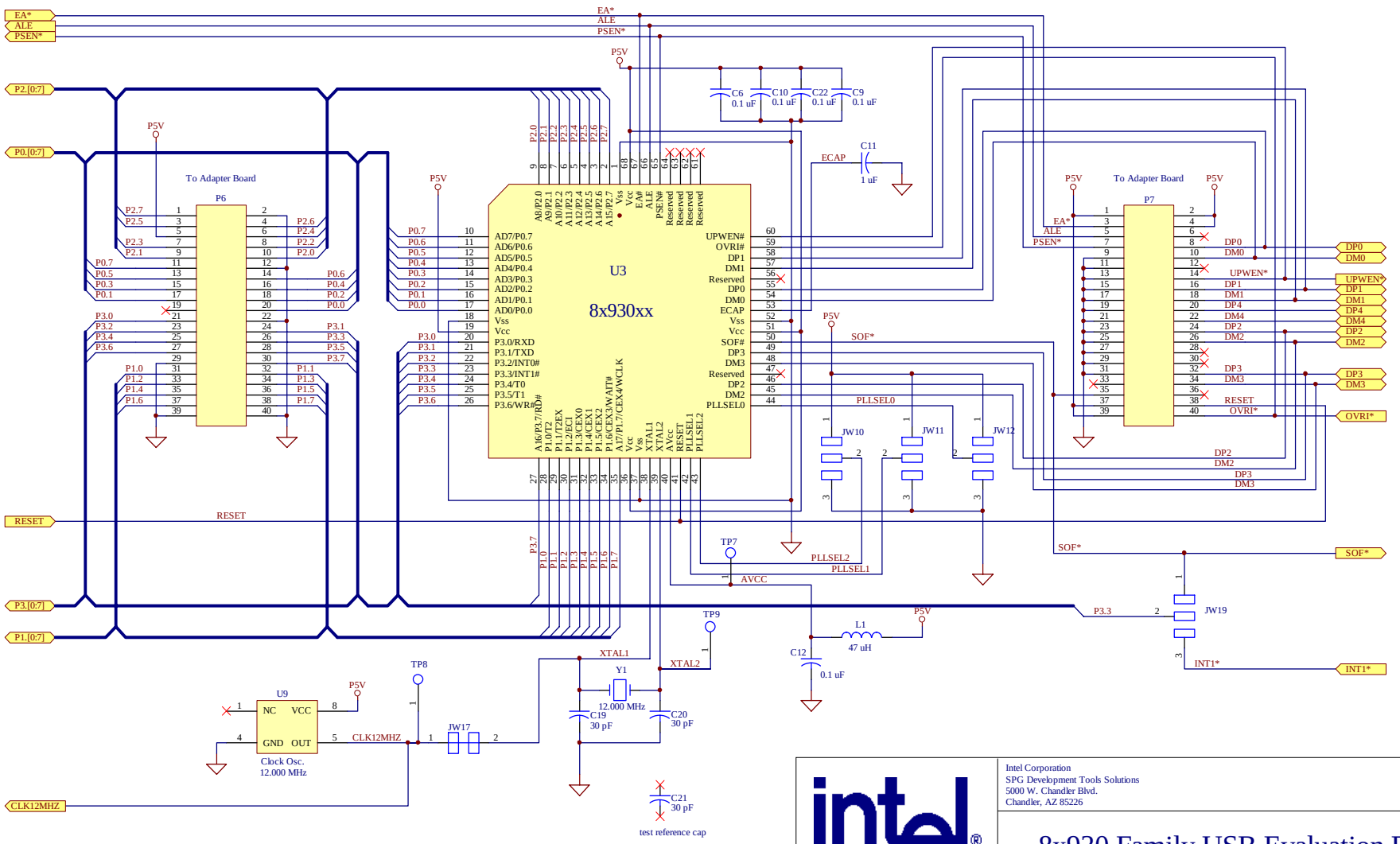
Sheet 2 usbcpu.sch
Sheet 3 usbepld.sch
Sheet 4 usbmem.sch
Sheet 5 usbuart.sch
Sheet 6 usbport.sch
Sheet 7 usbpwr.sch
Sheet 8 usbstat.sch
Sheet 9 usbiocon.sch

Intel Corporation
SPG Development Tools Solutions
5000 W. Chandler Blvd.
Chandler, AZ 85226

8x930 Family USB Evaluation Bd.

Size	FCSM No.	DWG No.	Rev
B		0001	B

File	Drawn by	Date	Sheet
C:\82930AA\NREVB\82930.PRJ	CMJ	10-Jan-1997 15:15:29	1 of 9



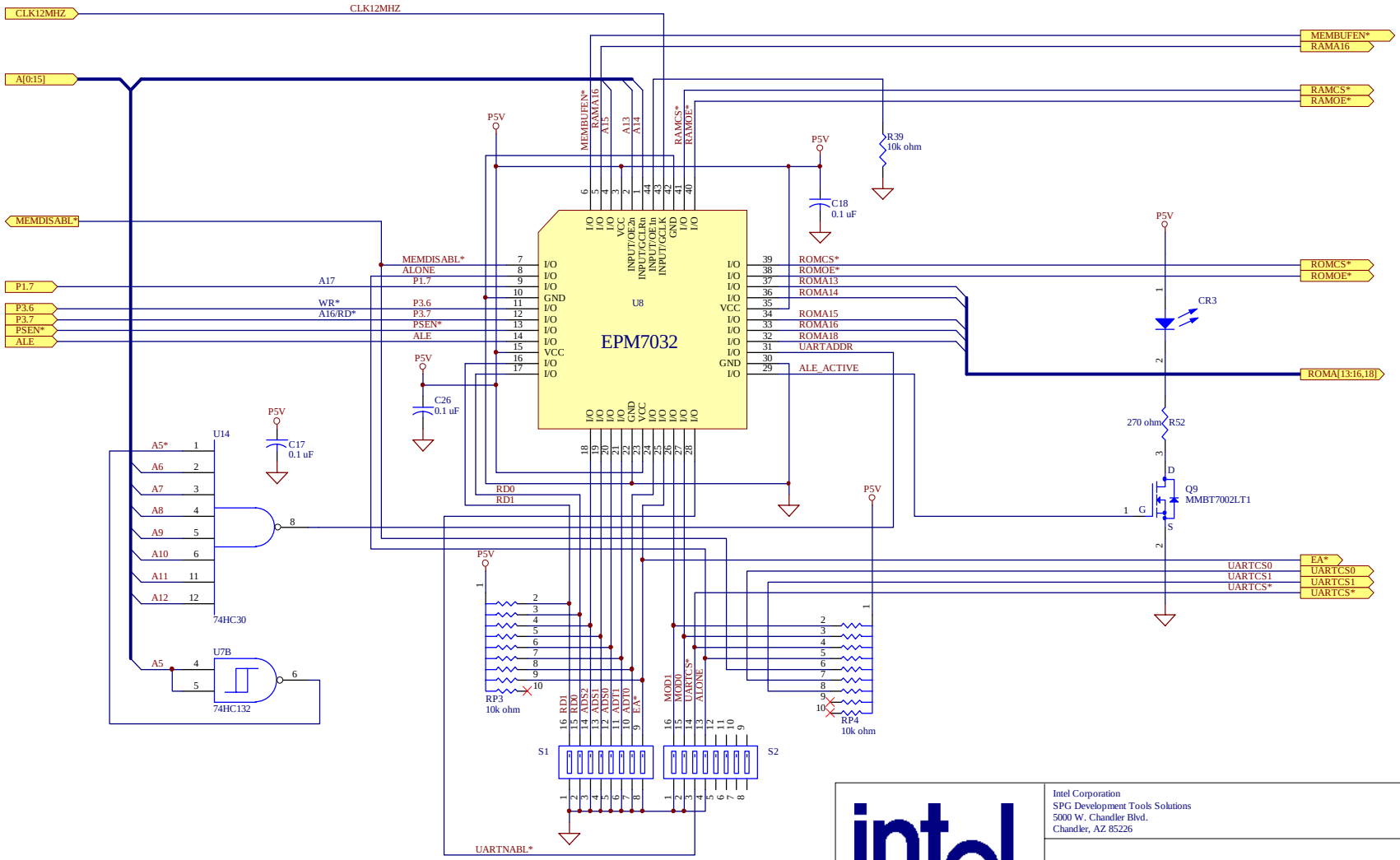
Although Intel has verified this design to be functional, Intel assumes no responsibility for any errors which may appear in the design. Intel reserves the right to modify this design without notice.

Intel Corporation
SPG Development Tools Solutions
5000 W. Chandler Blvd.
Chandler, AZ 85226

8x930 Family USB Evaluation Bd.

Size	FCSM No.	DWG No.	Rev
B		0001	B

File	Drawn by	Date	Sheet
C:\82930AA\NREVB\USBCPU.SCH	CMJ	10-Jan-1997 15:16:43	2 of 9



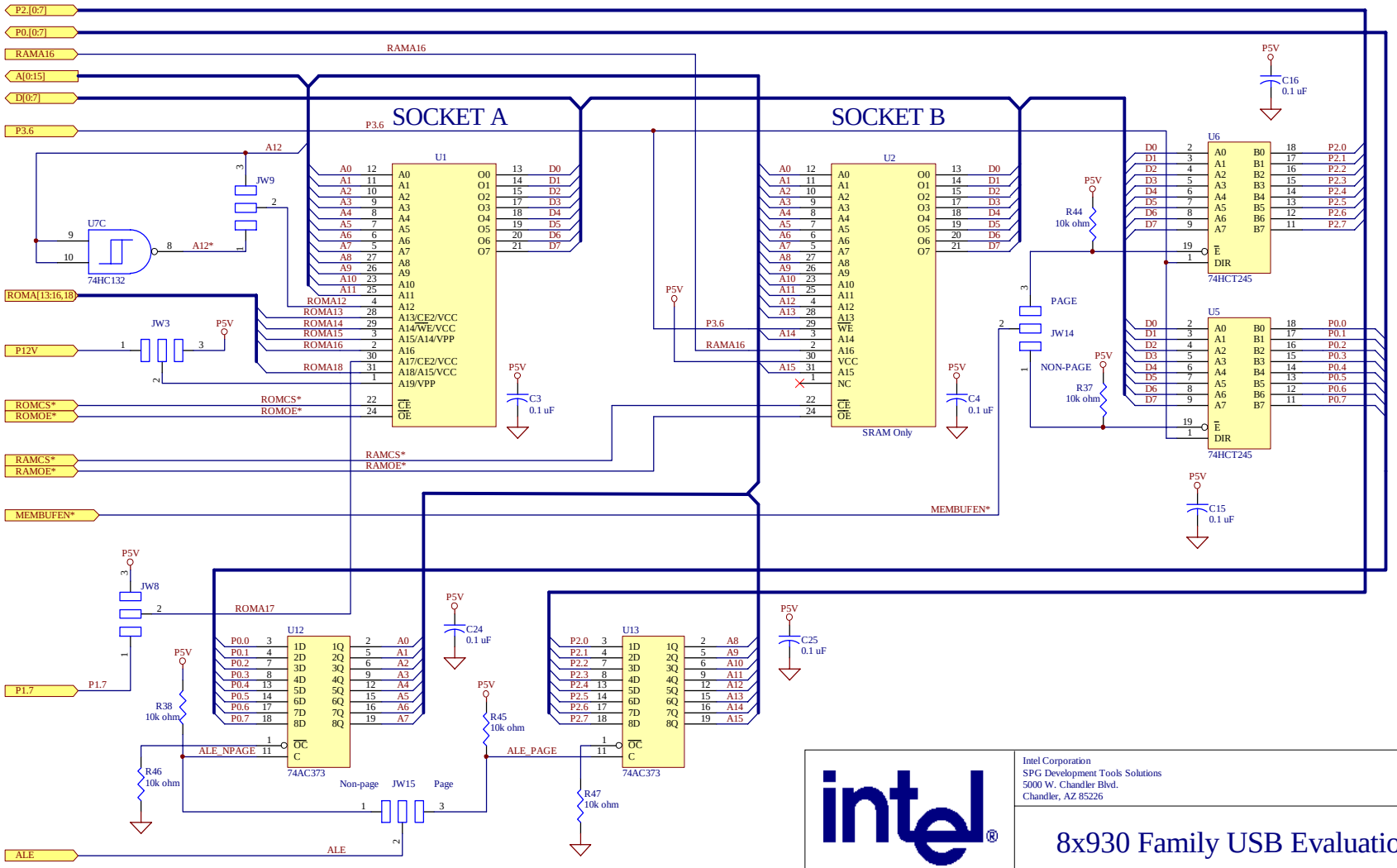
Although Intel has verified this design to be functional, Intel assumes no responsibility for any errors which may appear in the design. Intel reserves the right to modify this design without notice.

Intel Corporation
SPG Development Tools Solutions
5000 W. Chandler Blvd.
Chandler, AZ 85226

8x930 Family USB Evaluation Bd.

Size	FCSM No.	DWG No.	Rev
B		0001	B

File	Drawn by	Date	Sheet
C:\82930AA\NREVB\USBEPLD.SCH	CMJ	10-Jan-1997 15:18:16	3 of 9



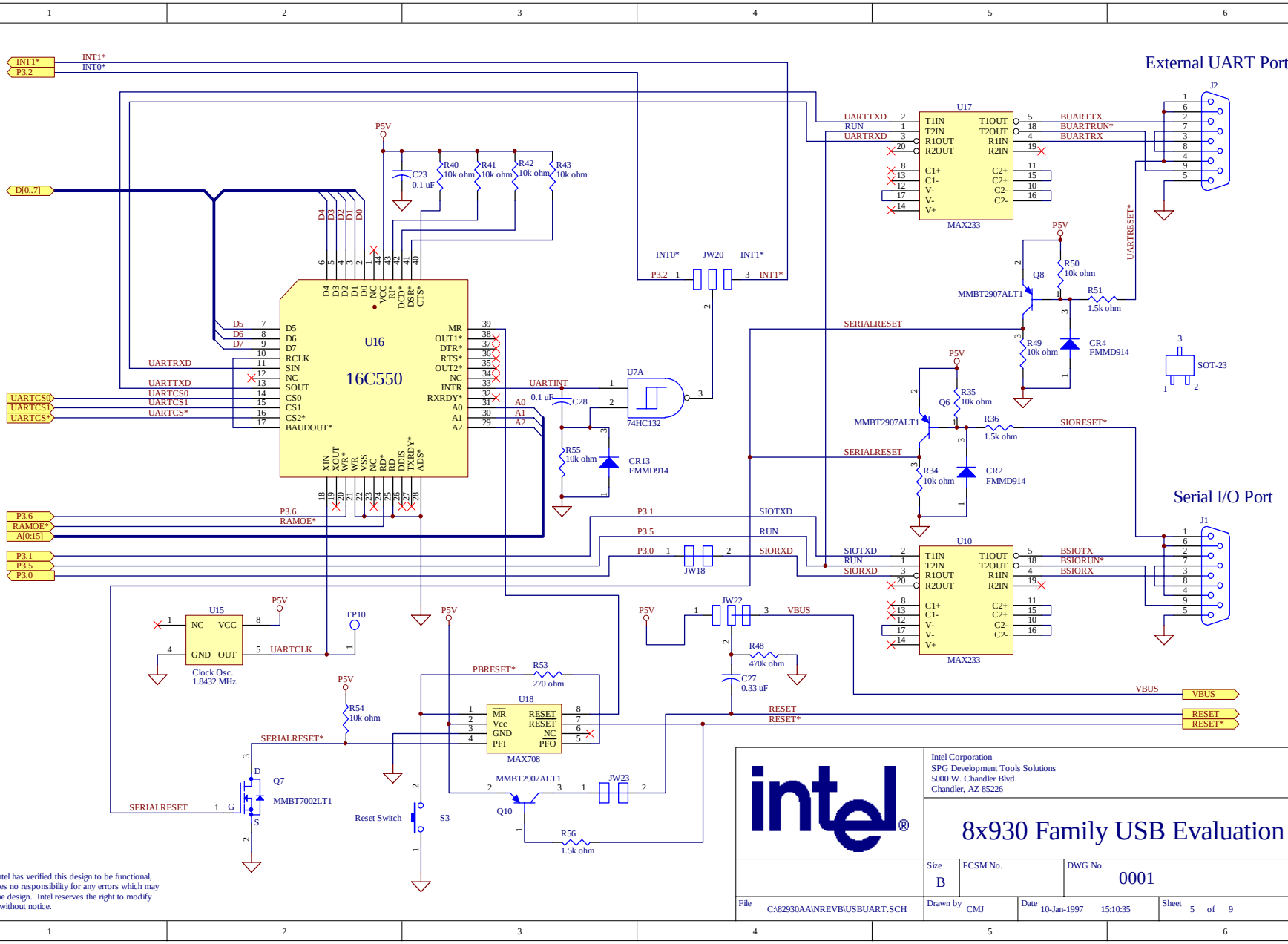
Although Intel has verified this design to be functional, Intel assumes no responsibility for any errors which may appear in the design. Intel reserves the right to modify this design without notice.

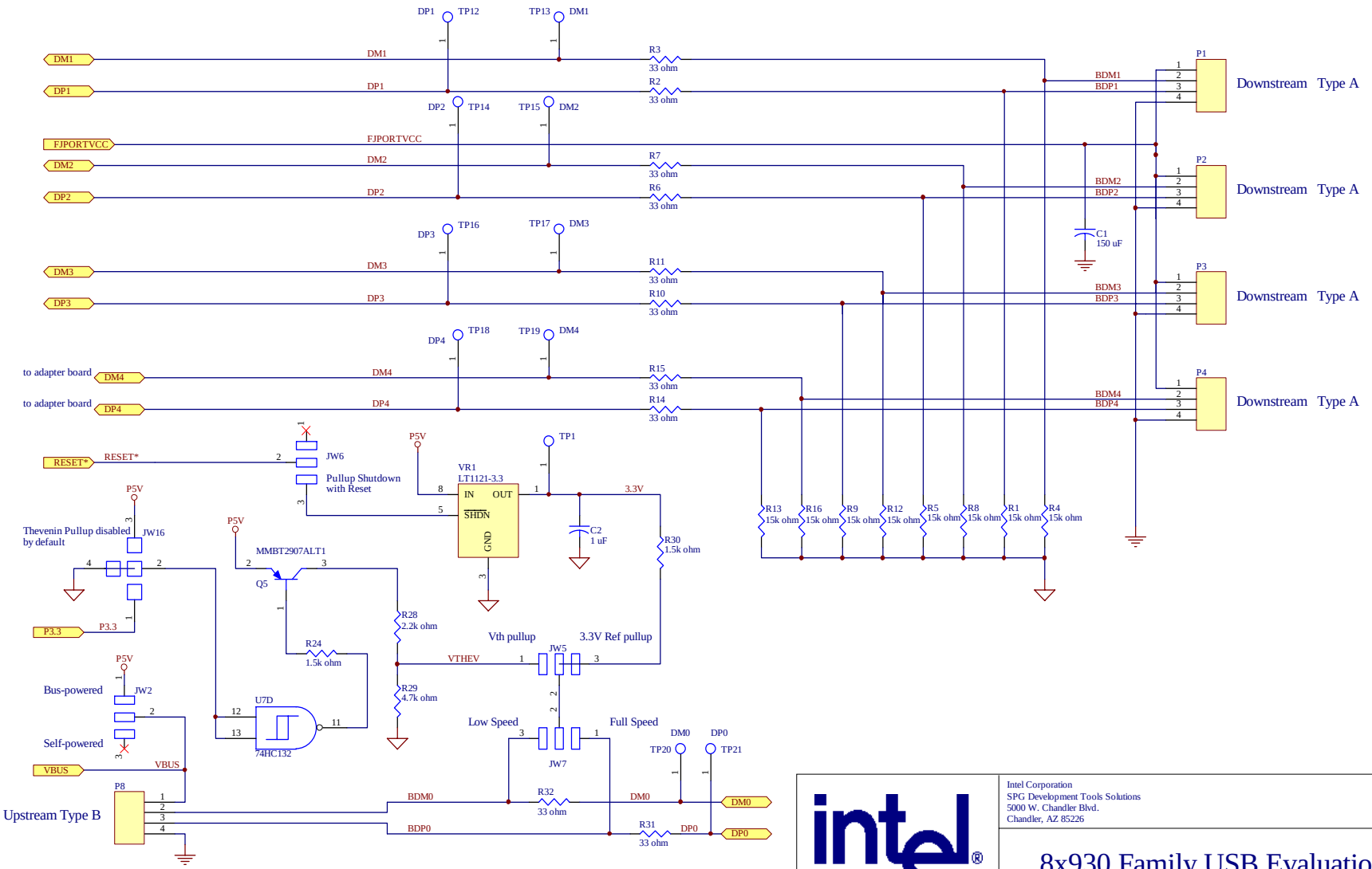
Intel Corporation
SPG Development Tools Solutions
5000 W. Chandler Blvd.
Chandler, AZ 85226

8x930 Family USB Evaluation Bd.

Size	FCSM No.	DWG No.	Rev
B		0001	B

File	Drawn by	Date	Sheet
C:\82930AA\NREVB\USBMEM.SCH	CMJ	10-Jan-1997 15:19:27	4 of 9


External UART Port
Serial I/O Port
U16
16C550
U17
MAX233
U10
MAX233
U15
Clock Osc.
1.8432 MHz
U18
MAX708
U7A
74HC132
J1
J2
SOT-23
Reset Switch
Intel Corporation
SPG Development Tools Solutions
5000 W. Chandler Blvd.
Chandler, AZ 85226
8x930 Family USB Evaluation Bd.
Size B
FCSM No.
DWG No. 0001
Rev B
File C:\82930AA\NREVB\USBUART.SCH
Drawn by CMJ
Date 10-Jan-1997 15:10:35
Sheet 5 of 9
Although Intel has verified this design to be functional, Intel assumes no responsibility for any errors which may appear in the design. Intel reserves the right to modify this design without notice.



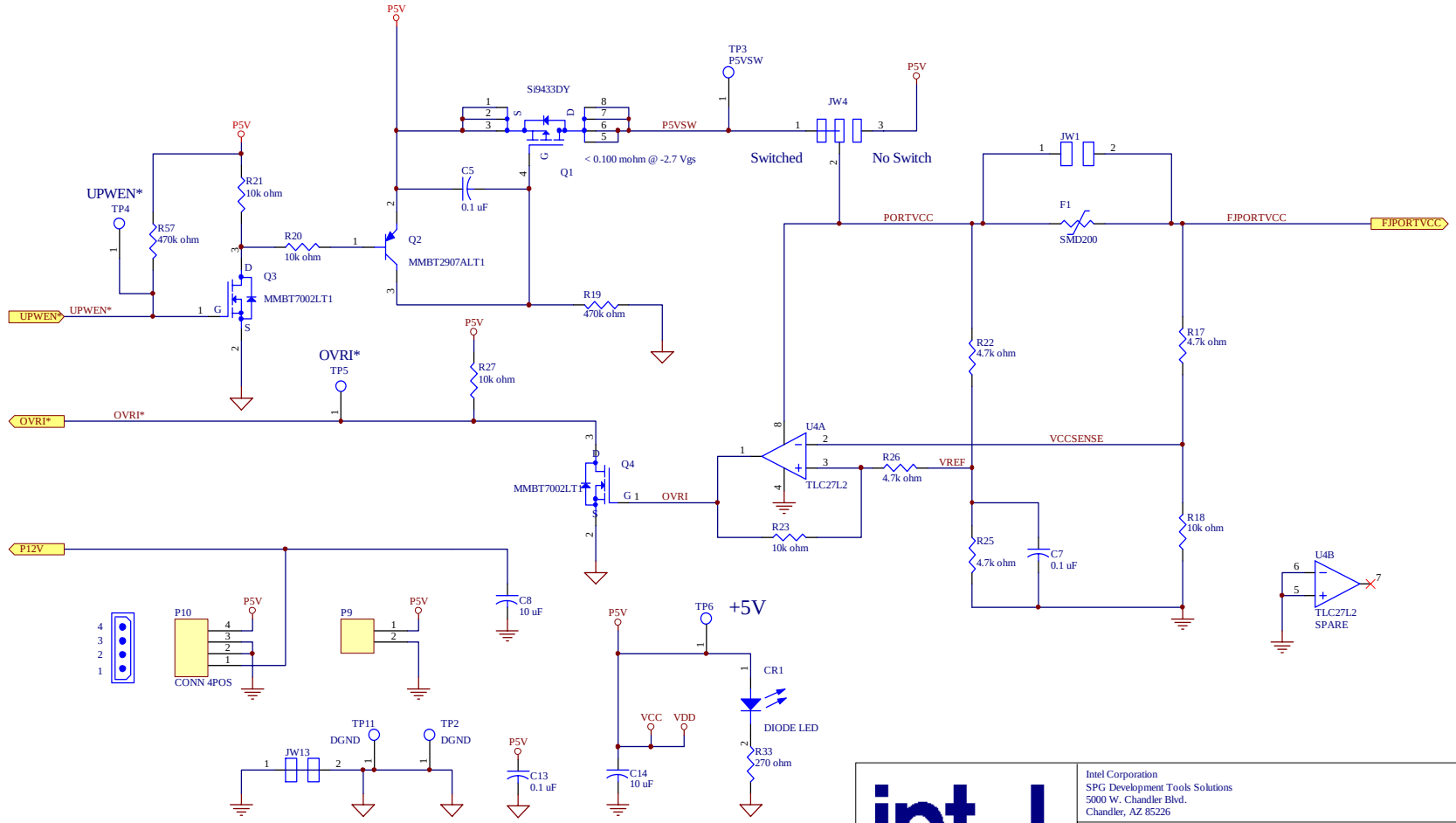
Although Intel has verified this design to be functional, Intel assumes no responsibility for any errors which may appear in the design. Intel reserves the right to modify this design without notice.

Intel Corporation
SPG Development Tools Solutions
5000 W. Chandler Blvd.
Chandler, AZ 85226

8x930 Family USB Evaluation Bd.

Size	FCSM No.	DWG No.	Rev
B		0001	B

File	Drawn by	Date	Sheet
C:\82930AA\NREVB\USBPORT.SCH	CMJ	10-Jan-1997 15:21:43	6 of 9



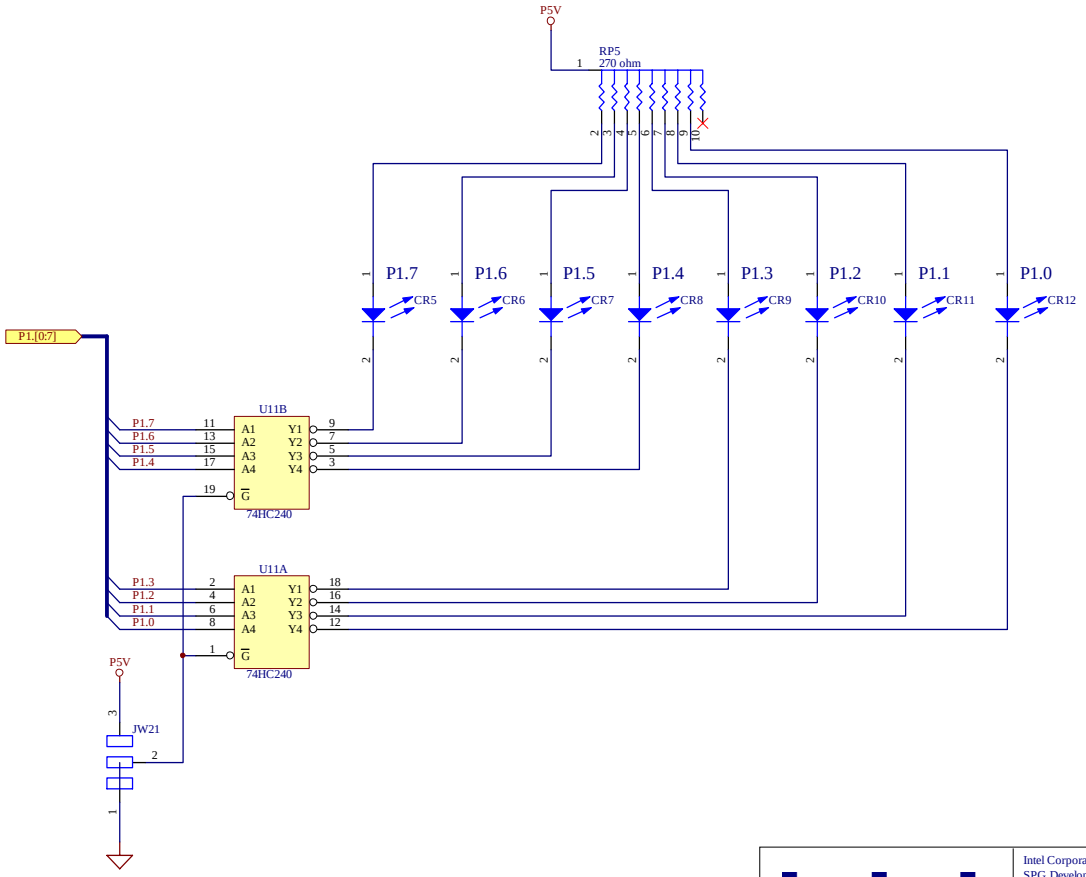
Although Intel has verified this design to be functional, Intel assumes no responsibility for any errors which may appear in the design. Intel reserves the right to modify this design without notice.

Intel Corporation
SPG Development Tools Solutions
5000 W. Chandler Blvd.
Chandler, AZ 85226

8x930 Family USB Evaluation Bd.

Size	FCSM No.	DWG No.	Rev
B		0001	B

File	Drawn by	Date	Sheet
C:\82930AA\NREVB\USBPWR.SCH	CMJ	10-Jan-1997 15:22:41	7 of 9



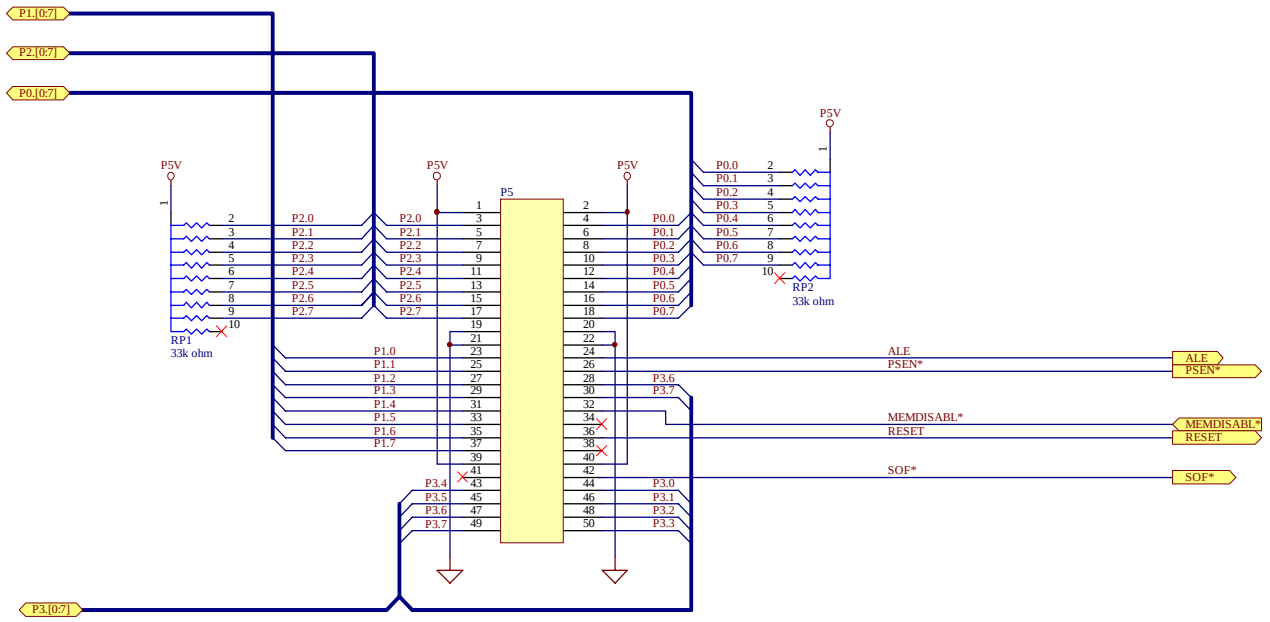
Although Intel has verified this design to be functional, Intel assumes no responsibility for any errors which may appear in the design. Intel reserves the right to modify this design without notice.

Intel Corporation
SPG Development Tools Solutions
5000 W. Chandler Blvd.
Chandler, AZ 85226

8x930 Family USB Evaluation Bd.

Size	FCSM No.	DWG No.	Rev
B		0001	B

File	Drawn by	Date	Sheet
C:\82930AA\NREVB\USBSTAT.SCH	CMJ	10-Jan-1997 15:25:11	8 of 9



Although Intel has verified this design to be functional, Intel assumes no responsibility for any errors which may appear in the design. Intel reserves the right to modify this design without notice.

Intel Corporation
SPG Development Tools Solutions
5000 W. Chandler Blvd.
Chandler, AZ 85226

8x930 Family USB Evaluation Bd.

Size	FCSM No.	DWG No.	Rev
B		0001	B

File	Drawn by	Date	Sheet
C:\82930AA\NREVB\USBIOCON.SCH	CMJ	10-Jan-1997 15:38:39	9 of 9

Index

M

N

P

R

S

T

U

W